Towards Cloud Based Architectures for Robotic Application Provisioning

Fatima Zahra Errounda

A Thesis in

The Department

of

Computer Science and Software Engineering

Presented in Partial Fulfillment of the Requirements

for the Degree of Master of Applied Science (Software Engineering)

at Concordia University

Montreal, Québec, Canada

December 2013

CONCORDIA UNIVERSITY
SCHOOL OF GRADUATE STUDIES

This is to certify that the thesis prepared

By: Fatima Zahra Errounda

Entitled: "Towards Cloud-Based Architectures for Robotic Applications Provisioning" and submitted in partial fulfillment of the requirements of the degree of

Master of Applied Science

Complies with the regulations of the University and meets the accepted standards with respect to originality and quality.

Signed by the final committee:

______________________________________Chair
Dr Peter Rigby

____________________________________ Examiner
Dr Todd Eavis

______________________________________Examiner
Dr Joey Paquet

______________________________________Supervisor
Dr. Roch. Glitho

Approved by: __________________________________
Dr. Sudhir Mudur
Department of Computer Science and Software Engineering

_______________20_____

____
Dr. Christopher Trueman
Dean, Faculty of Engineering and Computer Science

ABSTRACT

Robotic applications are widely used in various domains, such as healthcare, precision agriculture, and disaster management. However, provisioning them in a cost-efficient manner remains an uphill task. Cloud computing is changing the way applications are provisioned. It is a new paradigm with three key facets: Infrastructure as a Service (IaaS), Platform as a Service (PaaS) and Software as a Service (SaaS). Access anytime, anywhere, rapid application development through third party application re-use, and efficient use of resources are among cloud computing's expected benefits. All these expected benefits can potentially lead to cost efficiency in robotic applications provisioning.

This thesis defines a set of requirements for cost-efficient robotic application provisioning. It starts by reviewing the state of the art. A key issue from business perspective is how robotic application and third party application clouds could interact for cost efficient robotic applications provisioning purpose. This thesis proposes a business model that can potentially solve the issue. It is made up of four categories of actors: the end-users, the robotic cloud providers, the third party cloud providers and the cloud interaction framework provider. It is actually the cloud interaction framework provider that enables the interactions between robotic application clouds and third party application clouds. We also propose a peer to peer (P2P) overlay network as the basis of the architecture of the interaction framework provider role. It is a P2P overlay network in which each

cloud provider (i.e. robotic cloud and third party cloud providers) is represented by a node for publication, discovery, activation, and execution of applications purposes.

As a proof of concept, we implemented the P2P overlay based architecture including the P2P overlay network with its nodes. We have also implemented a robotic application (i.e., wildfire suppression) and the third party applications it re-uses (i.e., fire detection and supply management) with robotic application and third party applications residing in separate clouds. In addition, performance measurements have also been made.

Acknowledgments

I wish to express my gratitude and appreciation to Dr. Glitho and Dr. Belqasmi who helped me make this master thesis possible. I thank Dr. Glitho for his continuous assistance, support and insightful advices. I thank Dr. Belqasmi for her encouragement and kindness.

I would like to thank my colleagues at Concordia's lab for their help and suggestions. I would like to thank Carla Mouradian for her help and dedication.

I am grateful to Dr Eavis and Dr Paquet for serving as members of my thesis committee.

I am grateful to Concordia University for their financial support and for giving me the opportunity to work in research and follow my dreams.

I would like to thank my family and friends who have showed me a tremendous care and support.

Table of Contents

List of figures

List of tables

Acronyms and abbreviations

SLA	Service-Level Agreement
IaaS	Infrastructure as a Service
PaaS	Platform as a Service
SaaS	Software as a Service
P2P	Peer-to-Peer
ISO	International Organization for Standardization
MSRS	Microsoft Robotics Studio
VPL	Visual Programming Language
RTM	Robotic Technology Middleware
ROS	Robot Operating System
DHT	Distributed Hash Table
SOA	Service Oriented Architectures
WSDL	Web Service Definition Language
MRDS	Microsoft Robotic Development Studio
DSS	Decentralized Software Services
SOAP	Simple Object Access Protocol
ROS	Robotic Operating System

SLAM	Simultaneous Localization and Mapping
WOL	Web Ontology Language
URI	Uniform Resource Identifier
CAD	Computer-Aided Design
HTML	HyperText Markup Language
REST	Representational State Transfer
SCRS	Senior Companion Robot System
GAE	Google Apps Engine
URL	Uniform Resource Locator
HTTP	Hypertext Transfer Protocol

1 Introduction

This chapter starts with definitions, including cloud computing, robots, robotic applications, and application provisioning. Then, the problem statement and the motivations are presented. Next, it presents the thesis contributions. The last section introduces the thesis organization.

1.1 Definitions

1.1.1 Cloud computing

There are many definitions of cloud computing; a definition that we find holistic of cloud computing aspects is the one stated in [1]. It defines a cloud as a large pool of virtualized resources that are easy to access. These resources can be hardware, development platforms and/or services. Resources are used in an optimal way by dynamically reconfiguring them to adjust a variable load. Typically customers pay for exploiting these resources per usage. The application provider offers some guarantees that are agreed upon with the customer in a personalized Service-Level Agreement (SLA).

Cloud computing covers three main aspects of services, namely Infrastructure as a Service (IaaS), Platform as a Service (PaaS) and Software as a Service (SaaS). IaaS delivers computing infrastructure as a service, this infrastructure contains resources such as servers, routers, and memory. PaaS offers developers a platform with systems, and an environment that supports development, testing, deployment, and hosting of applications. SaaS provides multiple customers simultaneously with instances of a software application with its underlying resources [2]. Customers of IaaS, PaaS and SaaS may be end-users or other applications.

1.1.2 Robots

Several organizations have initiated vocabulary standardization in the robotic field [3]. The International Organization for Standardization (ISO) has published ISO 8373 in 1994; it defines vocabulary mainly related to industrial robots. The latest version of ISO 8373 includes a general definition of robots and new terms related to service robots.

In this thesis we used the latest revision of ISO 8373 to define a robot. The ISO 8373 defines a robot as follows: "actuated mechanism programmable in two or more axes with a degree of autonomy, moving within its environment, to perform intended tasks," [3]. ISO differentiates between industrial and service robots, by defining the former as automatically controlled mechanism with three axes at least that are dedicated to use in industrial automation applications, and defining service robots as robots that perform useful tasks for humans or society. ISO defines teleoperation as the remote control of a robot by a human in real time. ISO's robot definition covers intelligent robots and remote-controlled ones. It also covers ground robots, such as Roomba and aerial robots (e.g., unmanned aerial vehicles or drones). In this thesis we use the general robot definition as defined by ISO, which include all types of robots.

1.1.3 Robotic Applications

A robotic application is the software that enables control of the robots, it can be running on the robot infrastructure itself or on another system that is connected to the robot. There are many aspects of robotic applications one can study. We can categorize robotic applications into applications that allow remote control of the robots, and applications that allow robots to adopt different behaviours according to the information they get about the environment from sensors and actuators. We can also categorize robotic applications depending on their application domains, or the degree of cooperation among different robots. In this thesis, we focus on robotic applications at large and our results are applicable across domains.

1.1.4 Application Provisioning

Application provisioning life cycle includes application development, deployment, operation, and withdrawal [42]. The application provider is responsible for building up the application and setting it up for deployment and operation. For that purpose, developers may discover and compose third party applications. For an application provider, third party applications are applications that are provisioned by any other provider. Application deployment involves the installation and the activation of the application. For the application installation, it is necessary to select the infrastructure provider that will host the application, publish the application, and negotiate the Service Level of Agreement (SLA) terms between the application provider and the infrastructure provider. Application activation prepares the application for instantiation and usage. Application operation comprehends authorization management, execution and usage monitoring. Application withdrawal is concerned with the deactivation and removal of the application.

Fault, performance, quality, and security management are also part of the application provisioning, but not the focus of this thesis. In this thesis, we focus on the publication and discovery aspects of the development phase, the activation aspect of the deployment phase and the execution aspect of the operation phase.

1.2 Motivations and Problem Statement

Even though robots are increasingly present in various fields, provisioning robotic applications in cost-efficient manner remains an uphill task. Because of the diversity of robot vendors and the lack of standards interface technologies to operate these robots, it is not easy to re-use common software components to develop robotic applications. Also, robotic applications in general are not accessible anytime anywhere. Furthermore, efficient usage of robots sensors and actuators is generally not met; because a robot is usually designed to perform intended tasks, its sensors and actuators are not used during the time when the robot is idle.

Several benefits are expected from cloud computing. By offering robotic applications as services in clouds, it will guarantee access from anywhere and at any time. It will also offer the possibility to develop these robotic applications by re-using third party applications that are offered in different clouds through inter-cloud interaction. Resource efficiency can be achieved through the co-existence of applications on the same robotic infrastructure, hence maximizing usage of sensors and actuators. These benefits make cloud computing an ideal candidate for application provisioning.

1.3 Thesis Contributions

The thesis contributions are as follows:

- A set of requirements on cloud-based architectures for robotic application provisioning
- A review of the state of the art related to our research subject with an evaluation of the related work taking into account the previously identified set of requirements
- An overall business model to allow the interaction between robotic application and third party application clouds for cost efficient robotic application provisioning purpose
- A cloud-based architecture that allows robotic applications to interact with third party applications for publication, discovery, activation, and execution purpose.
- A prototype that implements a real world scenario using the proposed solution along with the performance measurements.

1.4 Thesis Organization

The rest of the thesis is organized as follows:

Chapter 2 sets the background concepts about to the key concepts used in this thesis.

Chapter 3 introduces an illustrative application domain and the requirements derived from it, followed by the state of the art presentation and review.

Chapter 4 describes the proposed overall business model that we use to build the architecture solution.

Chapter 5 presents the cloud-based architecture for robotic provisioning that we propose.

Chapter 6 describes the prototype we implemented, also the tools and frameworks that we used.

Chapter 7 summarizes the overall contributions and identifies the research directions.

2 Background Information on Cloud Computing, Robotic Application Provisioning and Overlay

This chapter presents the main topics that are relevant to this thesis research area. The introduced topics are: robotic applications, application provisioning, cloud computing, and P2P overlay networks. P2P overlay networks are the cornerstone for the cloud-based architectures for robotic application provisioning that we propose. We use P2P overlays because they allow easy sharing and discovery of information. Also, peers in the P2P overlay interact using a common interface technology. Moreover, P2P overlays are self-organized networks, as they do not require central management of joining and leaving peers.

2.1 Robotic Applications

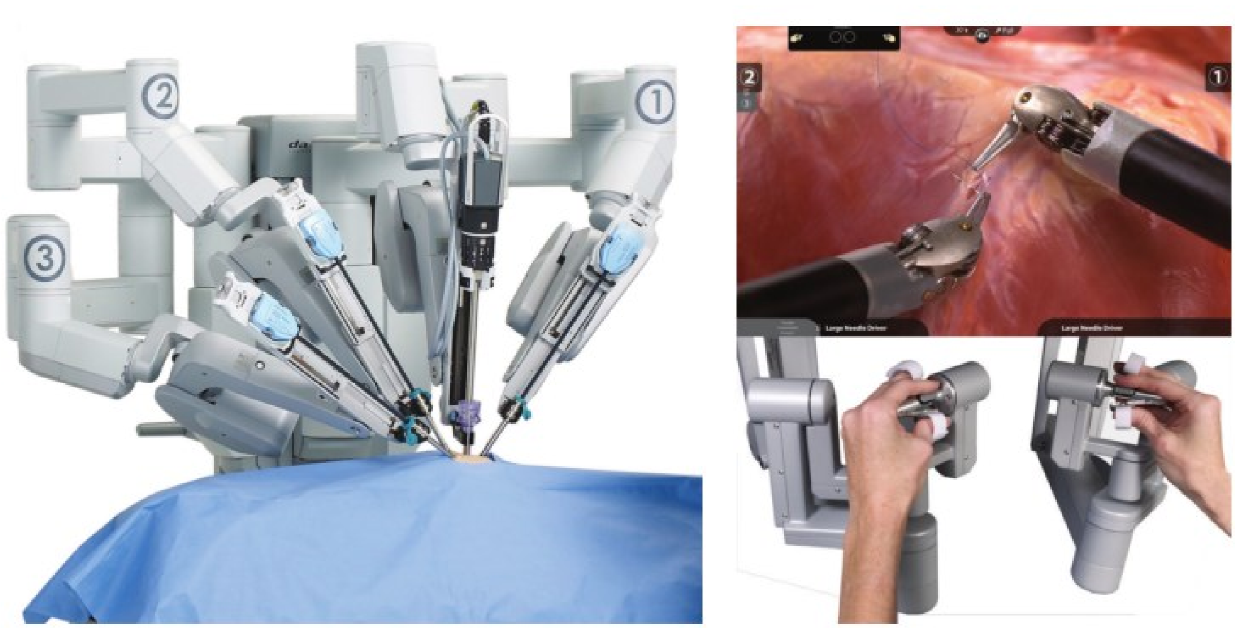

Figure 2-1: The Da Vinci system [12]

Robots are widely used in different domains. In healthcare, medical robots have emerged in surgery, patients' assistance, prosthetics and rehabilitation [12]. Medical robots were first used in

1985; a surgeon performed a brain biopsy using an industrial robot for defining the trajectory of the biopsy based on the computed images of the brain. In 2010, we counted a thousand Da Vinci systems installed worldwide [12]. The Da Vinci is a teleoperated system that allows surgeons to perform minimally invasive operations. Figure 2-1 shows the patient side and the controller side of the Da Vinci system. The use of medical robots has improved surgical outcomes, reduced patient trauma and shortened hospital stays [12].

Figure 2-2: Examples robots deployed as part of logistic system in hospitals [13]

Robots are also used in logistics systems for hospitals [13]. They carry goods that need to be transported, such as medicines, medical devices, specimens, food, documents, linen, and laundry. Robots in a hospital logistic system may operate on cooperative or supervisory modes. In cooperative mode, a fleet of robots work as a unit with a common mission. In supervisory mode, robots execute tasks that are scheduled and assigned by a centralized monitor [13]. Figure 2-2 shows examples of commercial systems that are successfully installed in hospitals.

The NASA Space Architecture Team uses robots for in-space operations including assembly, inspection, maintenance and planetary exploration [14]. During the 2004 Mars mission, for example, Mars Exploration Rovers (as illustrated in Figure 2-3), the robot that landed on Mars' surface, was guided by scientists and engineers to navigate the planet surface, manipulate instruments, and send images. NASA scientists can remotely operate in-space assembly robots arms to move and mate large components [14].

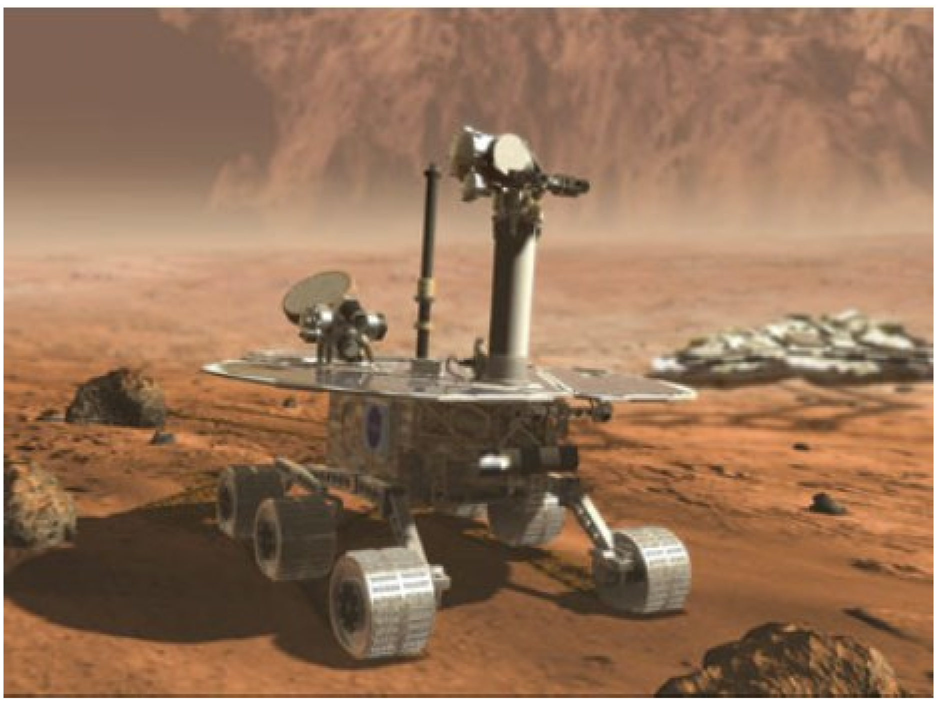

Figure 2-3: Mars exploration rovers [14]

Search and rescue robots play a vital role in disaster management operations [15]. They operate in regions that are dangerous or unreachable by human rescuers. Rescuers use the robots to collect information, carry radio transmitters, and bring food and medications to victims. Search and rescue robots can be categorized into aerial and ground robots. Aerial robots can maneuver in confined spaces that are difficult for a human being to reach.

They use cameras, sensors, and communication equipment to help the rescuer assess the situation. Ground robots are generally used for hazardous material operations, bomb disposal and military operations [15]. Figure 2-4 shows some examples of search and rescue robots.

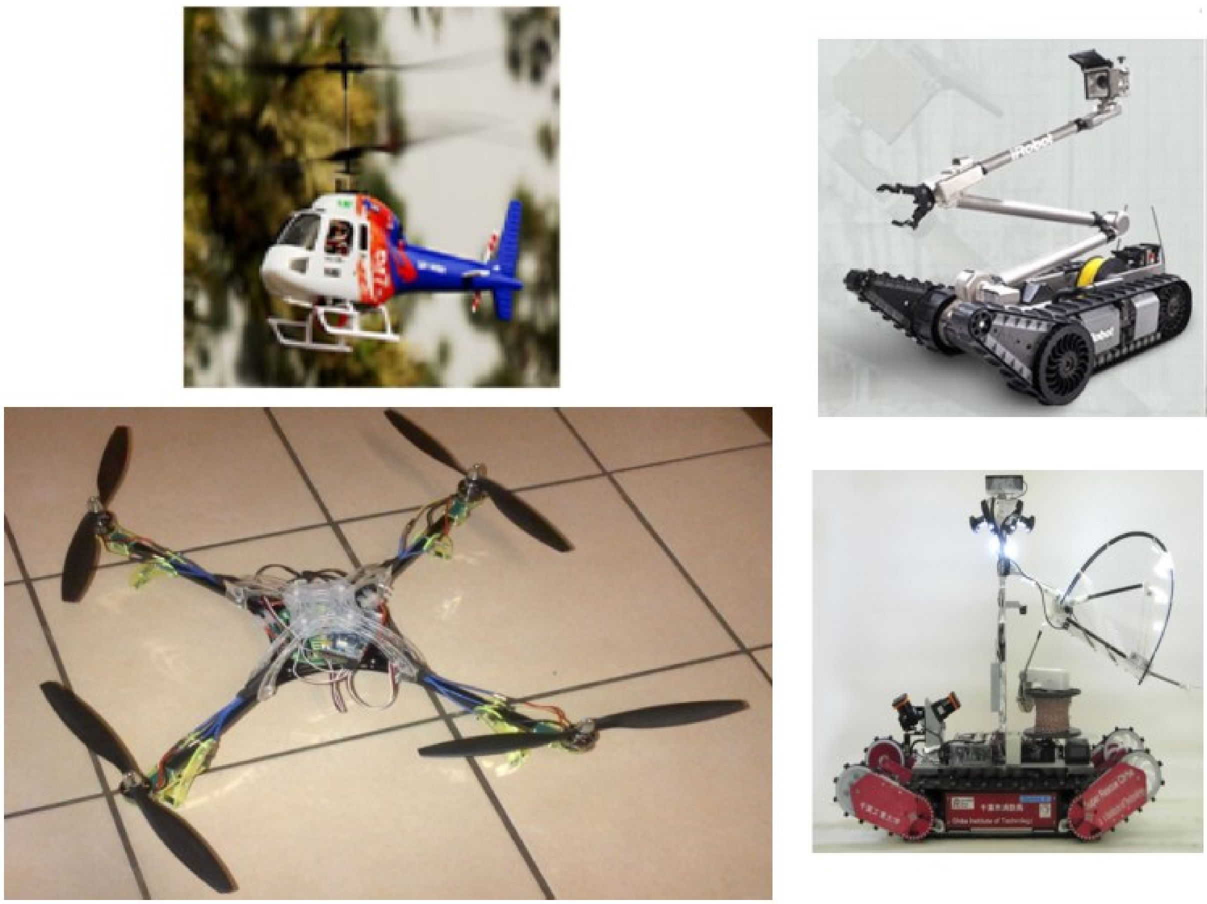

Figure 2-4: Search and rescue robots examples [15]

2.2 Application Provisioning

The overall lifecycle for application provisioning includes four phases: development, deployment, operation, and withdrawal [42]. The application lifecycle shows how application providers can create applications, make them available for customers, control their activation, and monitor their execution. Customers can be end-users or other applications. The application lifecycle must enable different applications to interact across different network domains in a distribution transparent way. The following sub-sections give an overview of these phases.

2.2.1 The Development Phase

The development phase includes the analysis, definition, specification, verification, implementation, validation, conformance testing, and system testing steps [42]. Applications are usually developed from an evolving environment of applications that each contribute to every step in the development phase.

The analysis step defines the requirements of each stakeholder involved in the application. A stakeholder could be the customer, other application providers, and the infrastructure provider. The definition step describes the functionality of the application and the synchronization needed with the other applications independently from any implementation. The specification step provides a formal description of the application logic, the synchronization with the other applications, and how that synchronization is achieved.

The verification step makes the correspondence between the specifications and the requirements. The implementation step includes the creation and re-use of the software components needed for the development of the application. These software components correspond to the applications identified in the previous steps. The validation step confronts the developed product to the requirements defined in the analysis step. The conformance step is checking if the implementation is following the architectural rules specified in the design that was determined in the definition and specification steps. The system testing step comprises the testing of the application in its operational environment.

In this thesis, we focus on mechanisms that allow the re-use of other applications to construct a given application. Such mechanisms are the publication and the discovery. We also focus on the loose coupling aspect between these applications. Loose coupling is the degree to which a relationship between two or more applications need to change when one of these applications change.

2.2.2 The Deployment Phase

The deployment phase has the installation and activation steps. The installation step is concerned with placing the software component into an operational environment with regards of compatibility and requirements to achieve the corresponding expected quality of service. The activation step puts the application in a state that allows customers to use it. In this thesis, we focus on the activation step.

2.2.3 The Operation Phase

The operation phase is concerned with the authorization management, execution, and usage monitoring. The authorization management step verifies that a customer is allowed to use the application. The execution step creates a link between the customer and the application through which they can interact, creates and invokes the necessary resources necessary for this interaction. Resources may be hardware components or other applications. The usage monitoring step collect resource usage information that can be used to apply the adequate charges for that usage and to verify the quality of service agreed upon in the SLA. In this thesis, we focus on the execution step of the operation phase, particularly in a heterogeneous, distributed environment.

2.2.4 The Withdrawal Phase

The withdrawal phase includes the deactivation and removal steps. The deactivation step is concerned with putting the application in a state that customers cannot use it anymore, unless it is re-activated. The removal step removes the software components from the operational environment. The withdrawal step is out of the scope of this thesis.

2.3 Cloud Computing

This sub-section present a general overview of cloud computing. We start with a brief definition of cloud computing. Then, we present the architectural and business models, followed by a sub-section that discusses the different cloud types. Finally, we explain the cloud computing characteristics.

2.3.1 Cloud Definition

The term "cloud" started gaining popularity after Google's CEO Eric Schmidt used the word to describe the business model for providing services across the Internet in 2006 [4]. However, due to the lack of a standard definition, the term cloud computing has been used in a variety of contexts to represent many different ideas, which generated confusion and skepticism about the cloud computing concept [4].

Several attempts of a standard definition of cloud computing have been proposed [1]. We adopted the definition of cloud computing as presented by The National Institute of Standards and Technology [4]; we find that it covers all the essential aspects of cloud computing:

"Cloud computing is a model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction" [4].

2.3.2 Cloud Architecture

Cloud computing can be represented using a service-driven business model, where hardware, middleware and applications are provided as on-demand services [4]. These services can be grouped into three layers: Software as a Service (SaaS), Platform as a Service (PaaS), and Infrastructure as a Service (IaaS). The upper layers build upon the capabilities provided by the lower layers. Every layer can be perceived as a customer of the layer below. Figure 2-5 illustrates the cloud computing layered architecture.

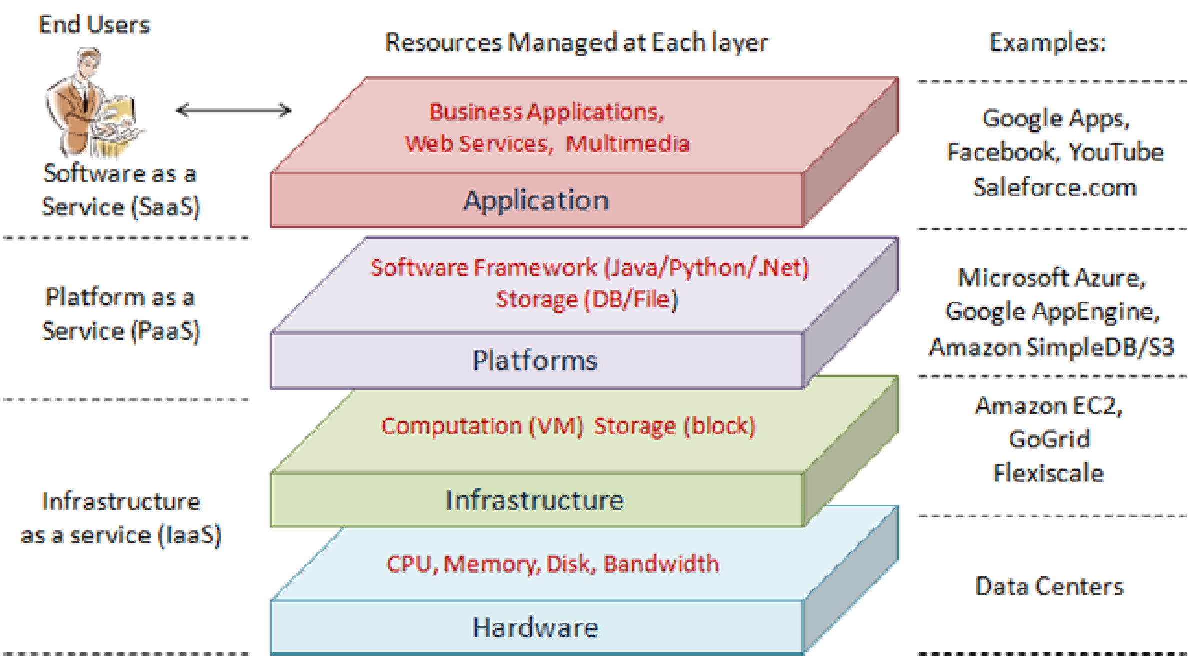

Figure 2-5: Cloud computing architecture [4]

Infrastructure as a Service: The IaaS layer is composed of physical and virtualized computing resources. These resources can be network infrastructures, storage or servers. Examples of IaaS providers include AmazonEC2 [5] and GoGrid [6].

Platform as a Service: The platform layer provides resources such as software development and management frameworks. They are used to develop, manage and deploy applications that will be provided as a service in the SaaS layer. Examples of PaaS providers include Google App Engine [7] and Microsoft Windows Azure [8].

Software as a Service: The SaaS layer provides on demand applications over the Internet. Cloud applications are different from traditional applications: their automatic-scaling feature helps them to achieve better performance, availability, and lower operating costs. Cloud applications can be accessed by end-users and other applications [4]. Examples of SaaS providers include Salesforce.com [9] and SAP Business ByDesign [10].

In this thesis, we focus on the PaaS layer and the development and deployment of applications.

2.3.3 Cloud Types

There are three different types of clouds: public, private, and hybrid [4]. These clouds differ in terms of operation costs, reliability, and security.

Public clouds: Public cloud providers offer their services to the general public [4]. Public clouds have several benefits. The absence of initial capital investment on infrastructure and risk shifting to infrastructure providers are among these benefits. However, users lack control over data, network, and security settings, which hinders public cloud use in some scenarios.

Private clouds: Private clouds are exclusively used by a single organization. They may be built and managed by the same organization or by external providers [4]. Private clouds give the users full control over data, network and security settings. But, they do not offer the benefit of no initial capital investments.

Hybrid clouds: A hybrid cloud is a mix of public and private clouds. It tries to tackle the limitations of each cloud type. In a hybrid cloud, some services run in private clouds, while the rest run in public clouds [4]. Hybrid clouds provide more control over resources than public clouds. At the same time, they allow service expansion and contraction in public clouds. However, designing the best distribution of services among public and private clouds is a complex task.

Service providers may select different cloud models depending on the business scenario. Public clouds, for instance, are suited for computation-intensive scientific applications because of their cost-effectiveness [4].

2.3.4 Cloud Computing Characteristics

Cloud computing provides several new features in contrast with traditional service computing [4]. The following characteristics are related to access of cloud services anywhere and anytime, rapid application development, and efficient use of resources:

Multi-tenancy: In a cloud environment, multiple providers may share a single data center to host their services. They may offer the same service to multiple tenants. The management of services is shared between the service provider and the infrastructure provider. Multi-tenancy enables cost saving in terms of licensing and infrastructure. But, multi-tenancy requires additional customization to maintain multiple tenants' profiles [4].

Shared resource pooling: Due to resource sharing, the infrastructure provider can assign resources to multiple customers dynamically. This dynamic assignment capability offers flexibility to the infrastructure providers in terms of resource management and operation costs [4]. Resource sharing is achieved through virtualization. Virtualization is a technology that allows the abstraction of computer resources by introducing a software abstraction layer between the hardware and the operating system or applications running on top of it [11].

Ubiquitous network access: Clouds services are mostly accessible through the Internet [4]. Therefore, even small foot print devices, such as mobile phones, PDAs, sensors, and potentially robots, with Internet connectivity are able to use these services.

Service orientation: As discussed above, cloud computing uses a service-driven model. Each IaaS, PaaS and SaaS provider negotiates its services with customers. The Service Level Agreement (SLA) is the result of that negotiation [4].

Self-organizing: Cloud customers can organize and manage their resources depending on their consumption needs, as resources can be allocated and freed on demand [4]. Freed resources can then be assigned to other customers without the need of a centralized management.

2.4 P2P Overlay Networks

In this sub-section, we start by giving a definition of the P2P overlay networks. Then, we describe the P2P overlay network architecture. Next, we present the overlay types, which are structured and unstructured networks.

2.4.1 Overlay Definition

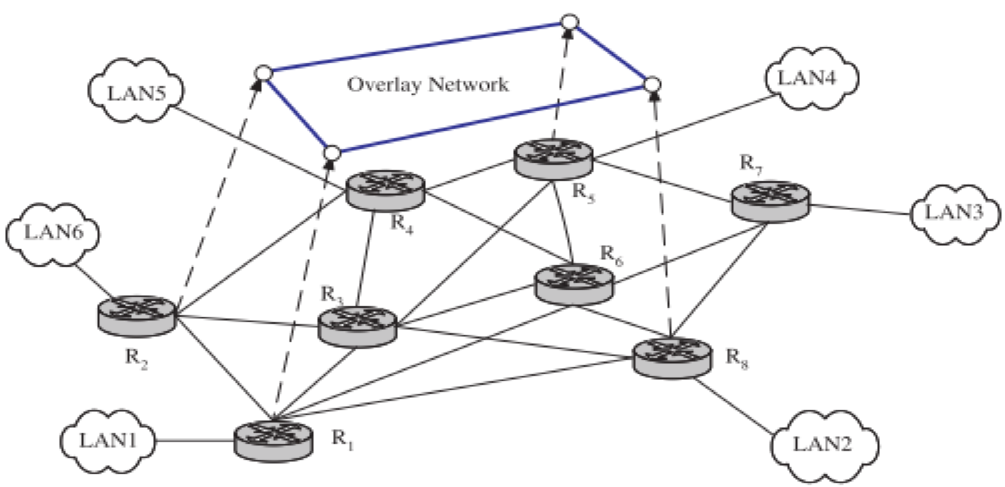

Figure 2-6: Overlay network example [19]

Overlay networks are virtual networks that are built on top of other networks. Nodes in the overlay can be connected by virtual or logical links that may override many physical links in the underlying network [19]. Figure 2-6 shows an example of an overlay network. Overlay networks require no changes to the underlying network, which makes them perfect candidates to share and discover services, and deploy new features and fixes in a cost-efficient manner.

2.4.2 P2P Overlay Network Architecture

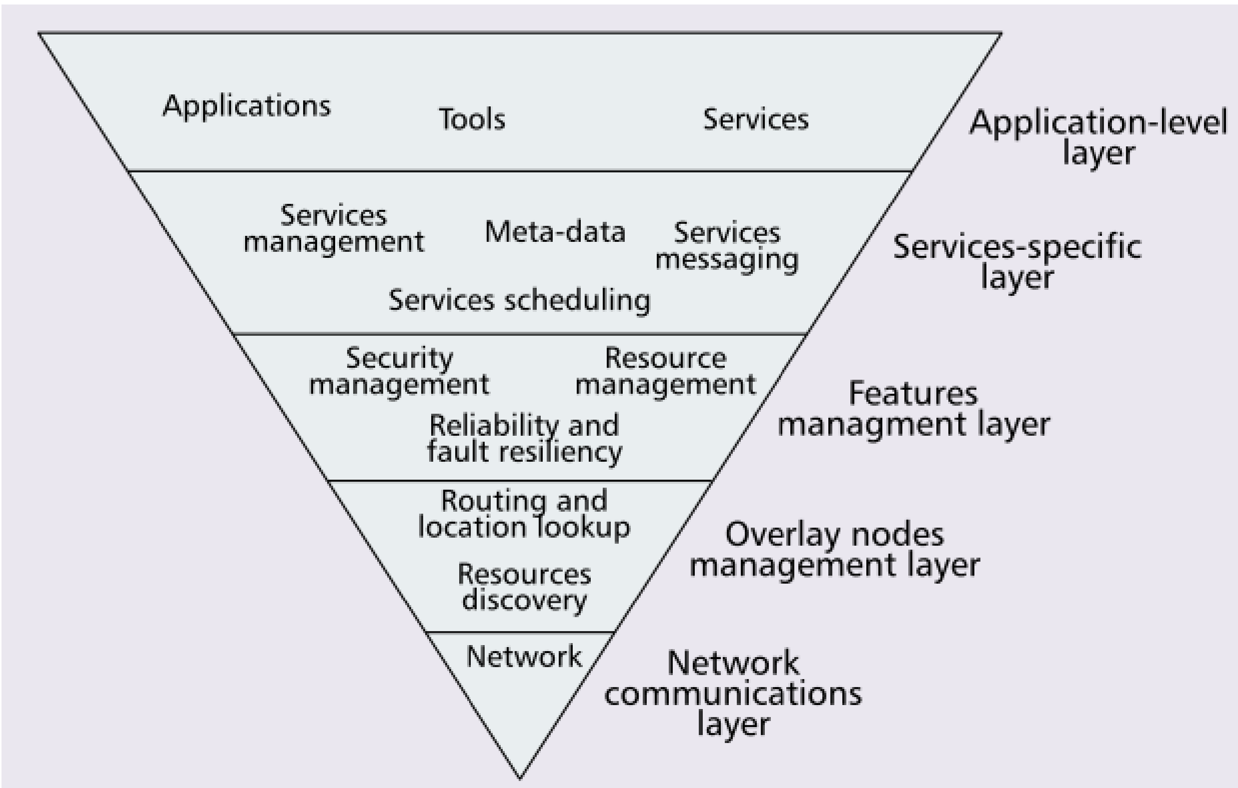

Figure 2-7: Overlay networks architecture [20]

A typical architecture of P2P overlay networks is illustrated in Figure 2-7 [20]; the architecture is composed of five layers. The network communication layer holds the network characteristics of the devices that compose the overlay. The overlay nodes management layer provides discovery and routing algorithms to manage the overlay peers. The features management layer is responsible for security, fault resiliency, and reliability. The services-specific layer supports the application-specific components by scheduling computation tasks and managing files on the underlying P2P infrastructure. The application-level layer supports tools, applications, and services that are implemented on top of the P2P infrastructure.

2.4.3 Overlay Types

Overlay networks can be categorized into two groups: structured and unstructured networks [20].

2.4.3.1 Structured Networks

Structured networks provide control over the network topology of the P2P overlay network. In structured networks, each resource holds a unique key and resources are distributed over the network peers according to a predefined scheme. This deterministic distribution enables resource discovery in a more efficient way.

Structured networks use Distributed Hash Tables (DHT) as a substrate. DHT contain the correspondence between each resource key and the ID of the peer that holds it. Each peer holds a routing table of its closest neighbors. This routing table is used in query lookup and message routing. There are several schemes for assigning IDs to peers and keys to resources. These schemes differ in network organization and routing strategies [20]. For illustration purpose we will discuss two P2P middleware for structured overlays: Chord and JXTA.

2.4.3.1.1 Chord

In Chord, peers' identifiers and resources' identifies belong to the same identifier space [20]. Resources and peers are assigned IDs and keys respectively based on the same hash function. Chord maps keys to IDs using consistent hashing. Consistent hashing is a particular type of hashing that requires a minimal change in the mapping of keys to IDs after a peer has joined or left the network. The identifier space is ordered in a ring model. A key k is assigned to the first peer whose ID is equal or follows k in the identifier space. When a peer n joins the network, it acquires certain keys that were previously assigned to its predecessor. The lookup is based on matching keys to IDs; a key k is passed around peers until it finds a pair of peers which IDs include k. Each peer keeps a routing table, called finger table, which contains a number of its successors. This table needs to be updated after a peer joins or leaves the network. For this, Chord runs a protocol on the background to maintain the peer's routing tables.

2.4.3.1.2 JXTA

JXTA was originally created by Sun Microsystems [36]. It's an open source project that aims at creating platform-independent P2P overlay networks. In JXTA, peers are assigned 128-bit identifiers. Peers may play two roles: edge or super peers. Edge peers are normal user peers, while super peers have the capability to enable other peers' communication with firewalls. Super peers can also play the role of coordinators between edge peers to reduce the inter-peer communication overhead and increase scalability; this type of peers is also called Rendezvous. Peers can form groups based on shared resources or security policies. Inter-peers communication is based on XML messages that are carried over pipes. Pipes are asynchronous, unreliable, virtual communication channels. They offer two communication modes: point-to-point and propagation. Propagation pipes involve communication in a one-to-many fashion. Peers, peer groups and pipes are described using XML files called advertisements. Each peer has a set of advertisements that describes its resources, such as the groups it belongs to and the communication pipes it offers [36].

To create platform-independent overlay networks, JXTA defines a stack of six XML-based protocols. The Peer Discovery Protocol enables peers to find other peers on the network. It is also used to find advertisements of peer groups or any other resources. The Peer Resolver Protocol allows peers to make generic queries to search within other peers, peer groups, pipes and other resources. The Rendezvous Protocol is used by peers to register to communication coordination services that rendezvous peers offer. The Peer Information Protocol allows for peers' status and availability retrieval. The Pipe Binding Protocol is responsible for setting communication pipes between peers. The Peer Endpoint Protocol is used to find routes between two peers. Peers that want to join JXTA network must implement a set of these protocols [36].

2.4.3.2 Unstructured Networks

In unstructured networks, peers join the network without prior knowledge of the topology. This type of network uses the flooding mechanism to propagate queries over the network within a specific scope. When a peer receives a flood request, it sends back the list of its resources [20]. We present one of the widely used unstructured networks: Gnutella.

2.4.3.2.1 Gnutella

Gnutella is a protocol for distributed search over flat topology of peer. It supports a client/server search paradigm, while using a decentralized model for document location and retrieval. Peers may join the network following loose rules; however, the placement of resources on peers is not based on any knowledge of the topology. To locate a resource, the requesting peer sends a lookup query to all its neighbours within a certain radius.

Flooding is very effective in finding highly replicated resources, but peers can easily become overloaded with requests. On the other hand, DHT-based networks are highly scalable and can locate rare resources, but they introduce significant overhead [20].

2.5 Chapter summary

This chapter has presented the background information related to the thesis. We presented some of the robotic application domains, like healthcare, space exploration, and rescue missions.

Next, we presented the definition of cloud computing. We also described the cloud computing architecture, which has three layers Software as a Service, Platform as a Service and Infrastructure as a Service. We introduced three cloud types: public, hybrid and private. Afterwards, we listed some of cloud computing characteristics.

Last, we presented a definition of overlay networks. We described typical overlay architecture. Then we presented the two types of overlays: structured and unstructured. We also introduced some of the existing structured networks, such as Chord and JXTA. Finally we presented the most commonly used unstructured network: Gnutella.

3 Robotic Application Domain Description, Requirements, and the State of the Art

This chapter is composed of three sections. First, we describe a robotic application domain from which we derive requirements. Afterwards, we overview and evaluate the state of the art related to non-cloud robotic application provisioning and cloud-based robotic application provisioning solutions. Finally, we summarize the chapter.

3.1 Illustrative Robotic Application Domain

The wildfire suppression robotic application detects and suppresses wildfires using a fleet of heterogeneous robots. These robots can be aerial or ground firefighters. Aerial firefighters apply retardant to reduce fire intensity and spread rate while ground firefighters suppress fire using water and foam.

The robotic application can be accessed from anywhere, anytime, using a computer, a smartphone or a tablet. If a fire is detected, the application evaluates its intensity and its spread rate. Then it deploys the appropriate robots to extinguish it.

The wildfire suppression application re-uses a fire detection application and a supply management application. The fire detection application could utilize a network of sensors that are deployed in the forest to gather information about wind speed, gas concentration, moisture and temperature. The supply management application manages the fire retardant supply stations that firefighter robots use during fire suppression. These supply stations differ in terms of fire retardant type (i.e., water, foam) and accessibility to aerial or ground robots.

3.2 Requirements

We use the application provisioning lifecycle as a guide map to derive the requirements. We derive requirements for each phase, except for the withdrawal phase, which is out of the scope of the thesis. We also define general requirements that cover more than one phase in the provisioning lifecycle. This sub-section contains four sets of requirements: general requirements, development requirements, deployment requirements, and operation requirements. The following requirements are derived from the wildfire suppression application domain presented in the previous sub-section considering the focus of the thesis. We refer to third party applications as the applications re-used by a given application and that are provisioned by a different provider, they may be robotic or non-robotic applications.

3.2.1 General Requirements

One of the requirements is the necessity for third party applications to be offered through a standard web program - to program interface technology. This will enable the interactions required for re-using applications pertaining to different providers as building blocks to create other applications. It will also help in minimizing the number of technologies used in the design. For example, the fire detection and supply management applications may use different Web interfaces technologies (e.g., SOAP based – Web services or RESTful Web services).

3.2.2 Development Requirements

For the development phase, we focus on the analysis and the definition steps, especially on mechanisms that allow re-use of applications as building blocks to develop new applications. We do not cover the specification, verification, implementation, validation, conformance testing, and system testing steps.

The development-related requirement is the need for third party applications to be published and discovered as building blocks for robotic application providers to reuse while creating their applications. Indeed, the fire detection and the supply management application providers should be able to publish their applications. The wildfire suppression application provider must discover these applications to decide which application providers will be involved in the wildfire suppression application creation. It must also decide on the synchronization needed with the fire detection and the supply management application. In terms of synchronization, the fire detection provider will be involved first to alert the wildfire suppression application of a fire break out. Then, the wildfire suppression application uses the supply management application to determine the most adequate supply station to send the robots to for supply provisioning.

3.2.3 Deployment Requirements

For the deployment phase, we focus on the activation step. The installation step is out of the scope of this thesis. The activation-related requirement is the necessity for third party applications to be activated before their first use by the robotic applications; the robotic applications are responsible for requesting the third party applications activation. For example, the wildfire suppression application should be able to request the activation of the supply management application before its first use to determine the most adequate supply station.

3.2.4 Operation Requirements

For the operation phase, we focus on the execution step of the operation phase, particularly in a heterogeneous, distributed environment. We do not cover the authorization management and usage monitoring steps.

The first requirement is the necessity for robotic applications to request the execution of the third party applications. In the wildfire suppression application domain, the wildfire suppression application should be able to request the execution of the supply management and the fire detection applications.

The second requirement is the necessity to access the robotic applications anywhere and anytime for usage purpose. In the wildfire suppression application domain, the robotic application should be accessed using any device with Internet connectivity and it is accessible from anywhere and anytime to allow a real-time detection of the wildfires.

3.3 State of the Art Review

Cloud computing is a promising candidate for robotic application provisioning. Cloud computing characteristics can be used to satisfy virtually all requirements stated above. For example, cloud computing offers the possibility for applications to be accessed anywhere, anytime for operation purpose, using small footprint devices. Its multi-tenancy characteristic allows multiple third party applications providers to publish their applications to multiple tenants.

Cloud computing can be used in the previous application domain in two different ways. In the first approach robotic applications are offered as SaaS in robotic clouds. In the second one, third applications offered as SaaS are re-used by robotic cloud providers to build robotic applications.

In this sub-section, we organize the state of the art into two categories: non-cloud application provisioning solutions and cloud-based robotic application provisioning architectures. The cloud-based robotic application provisioning architecture is organized in three sub-categories: robotic applications that are offered as SaaS, robotic applications as cloud third party applications users, and inter-cloud architectures for application provisioning.

3.3.1 Non-Cloud Robotic Application Provisioning Solutions

In this sub-section we present the two most representative solutions. They both cover the development, the deployment, and the execution phases of the application provisioning lifecycle.

3.3.1.1 Microsoft Robotics Studio (MRSR)

MRSR was designed to become a standard in robot control software solutions [17]. In terms of development platform, MSRS provides a visual programming language (VPL) for novice developers to develop applications and for experienced developers to rapidly prototype their design. In VPL, each application can be expressed as a state machine. The interaction between the application and the robots is achieved through services that drive and monitor the robots' sensors and actuators. These applications and services can be deployed on the robot itself or on an intermediate computer that controls the robot, depending on the processing power of the robot. In terms of operation, the sensors and actuators' services are decoupled from the hardware components of the robots using a mapping layer. MRSR supports an XML-based distributed message technology to allow the interactions among the services. It offers the possibility to use Simple Object Access Protocol (SOAP) to execute the robots services; however service performance drops due to additional message encoding/decoding.

Even though MRSR allows the re-use of applications as building blocks to create new applications, it does not provide any publication and discovery mechanism; the application provider must know beforehand the services it can re-use. Moreover, it does not allow application activation. When it comes to operation requirements, although MRSR gives the possibility to access the applications as web services for execution, allowing access anytime and from anywhere for consumption purpose, it comes with the limitation that all services must run on a .NET environment, which ties the solution to a unique technology.

3.3.1.2 Robot Operating System (ROS)

ROS is not an operating system in terms of process management and scheduling; instead, it considers every process as a node connected to other nodes on P2P topology. These nodes can be hosted on heterogeneous operating systems [18]. It offers a communication layer that allows the interaction between these nodes. ROS is free and open source. In terms of development, it offers the possibility to build systems that consists of different nodes connected on a P2P topology. These nodes communicate using XML-based messages. An application in ROS is a cluster of interconnected nodes that communicate with each other. Nodes exchange messages based on a publish/subscribe model. ROS supports many programming languages and defines an interface language to describe messages that are exchanged between nodes. For the deployment phase, ROS was designed to run on an environment with different hosts connected at runtime. In terms of execution, the communication between two nodes is carried along pipelines.

Even though ROS allows the publication and discovery of nodes that can be used as building blocks to create applications, this publication and discovery can only be done during the operation phase. Furthermore, only nodes can be published, applications (clusters of nodes) cannot. ROS does not address activation. For the operation purpose, ROS does not allow access to applications from anywhere and anytime for consumption purpose.

3.3.2 Cloud-Based Robotic Application Provisioning Architectures

The first sub-category reviews the research work done to make robotic applications available as SaaS in robotic clouds. The second sub-category reviews the work done to build robotic application based on cloud applications. The third sub-category explores work done in inter-cloud architectures, as it might help in solving some of the issues related to the interactions between robotic applications that are offered as SaaS and the cloud third party applications these robotic applications re-use. To the best of our knowledge, there is no work related to robotic cloud interactions or inter-cloud interactions focusing on robots. Therefore, the work presented in the third category related to inter-cloud architectures for service provisioning at large.

3.3.2.1 Robotic Applications Offered as SaaS

There are several works that attempt to offer robotic applications as cloud applications. We present here two of these works that cover the development, deployment, and operation phases of the application provisioning lifecycle. Robot as a Service [21] proposes a novel architecture to offer robotic applications as a cloud application. Design of a Robot Cloud Center [23] builds on the first one and improves some of its aspects.

3.3.2.1.1 Robot as a Service

For the development phase, Chen's work [21] offers the possibility to re-use robots' sensing and actuation applications to develop new robotic applications. As depicted in Figure 3-1, each physical robot offers two types of robotic applications: simple and elaborated ones. Simple applications include sensor data collection and actuator data updates, while elaborate applications are more complex, such as maze navigation applications. Both types of applications are offered as SOAP-based web service. To decouple between these applications, Service Oriented Architecture (SOA) principles are used. SOA considers software resources as applications, which are well-defined, self-contained modules. These applications are described using Web Service Definition Language (WSDL). The separation of application interfaces and internal implementation gives applications their loosely coupled property [22].

SOA supports three key roles: an application provider, an application requester and an application broker. The application broker plays the role of an intermediary between applications requesters and applications providers [22]. Each robot holds a repository of all the applications that it offers, including simple and elaborated ones. It also plays the role of an application provider as well as an application broker. The robotic platform that was used is MRSR, which was reviewed in the previous section.

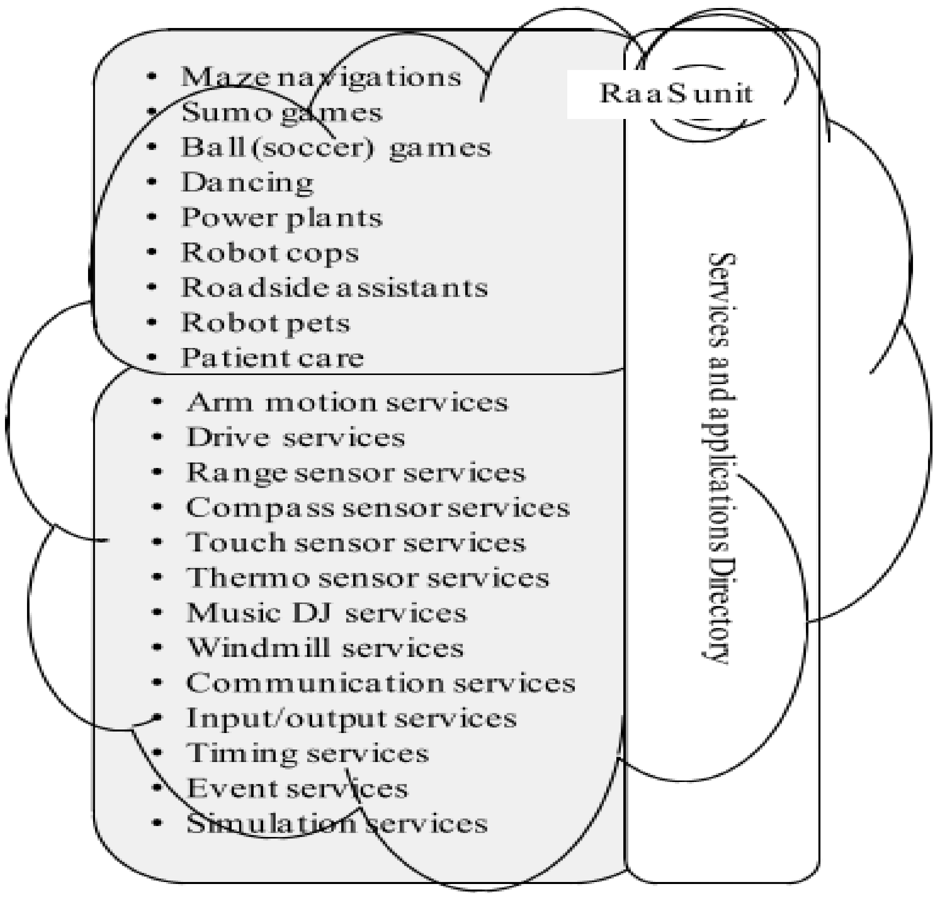

Figure 3-1: Robots as a service unit [21]

Although this work offers the possibility to re-use applications as building blocks to create new robotic applications, it is limited to one robot. It did not address the possibility to create new robotic applications on a robot using applications pertaining to other robots. Also, for the deployment phase, the activation of third party applications was not addressed. For the operation phase, the work is based on web services, but access anytime and anywhere for consumption purpose was not addressed.

3.3.2.1.2 Design of a Robot Cloud Center

Chen et al. enhanced some aspects of the previous work in [23]. At the development phase, the development platform remained MRDS. Additionally, development and deployment agents were developed to manage the assembling and deployment of applications as shown in Figure 3-2. The model panel gathers and analyzes user requirements. The assemble panel is used to compose new robotic applications. The deploy panel holds an auto-installer service to update new services running on the robots. And the management and analysis panel manages adding and substituting robots and also application brokers. They also moved the directory of robotic applications from the unit level to the network level.

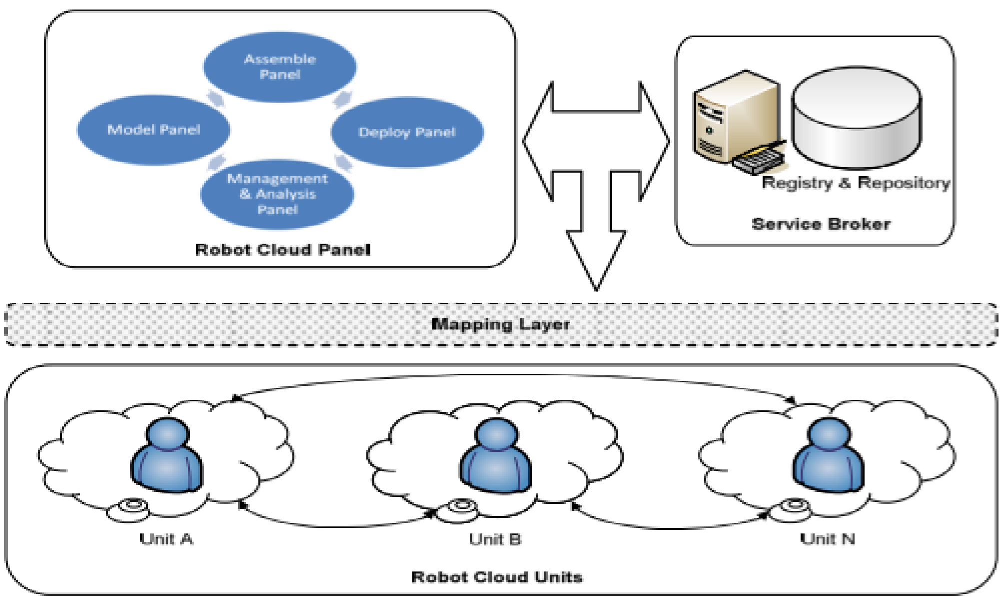

Figure 3-2: The robotic cloud architecture [23]

These changes gave the possibility to build robotic applications for a robot using applications pertaining to other robots. For the execution phase, a mapping layer was added on top of the robots' infrastructure to allow matching between the sensing and actuation services and the robots' sensing and actuation capabilities.

Although this work added the possibility to create new robotic applications based on existing applications, these applications are limited to the ones deployed on robots. Furthermore, it does not address the activation of these applications. For the operation phase and the interface technology, this work presents the same limitations as in the one it builds upon (i.e., [21]).

3.3.2.2 Robotic Applications as Cloud Third Party Applications Users

In this sub-section we overview works where robotic applications use third party applications for provisioning. Most of the works presented cover the development, deployment, and the operation phases of the application provisioning lifecycle.

First, we start with the DAvinCi [24] framework; it allows robotic applications to use a third party application to process data. Next, we present RoboEarth [26] which offers robotic applications a knowledge base where they can share and learn new skills. Then, we introduce the senior companion robot system [27]; it proposes an interactive system in which a robotic application interprets speech semantics using cloud applications. Next, we present a cloud-based approach to solve robot vision tasks using a smart camera system [28]. Finally, we present a simulation environment for robotic applications using cloud services [29].

3.3.2.2.1 DAvinCi

DAvinCi [24] is a software framework intended for the use of data-intensive distributed applications offered as SaaS in the cloud by robotic applications.

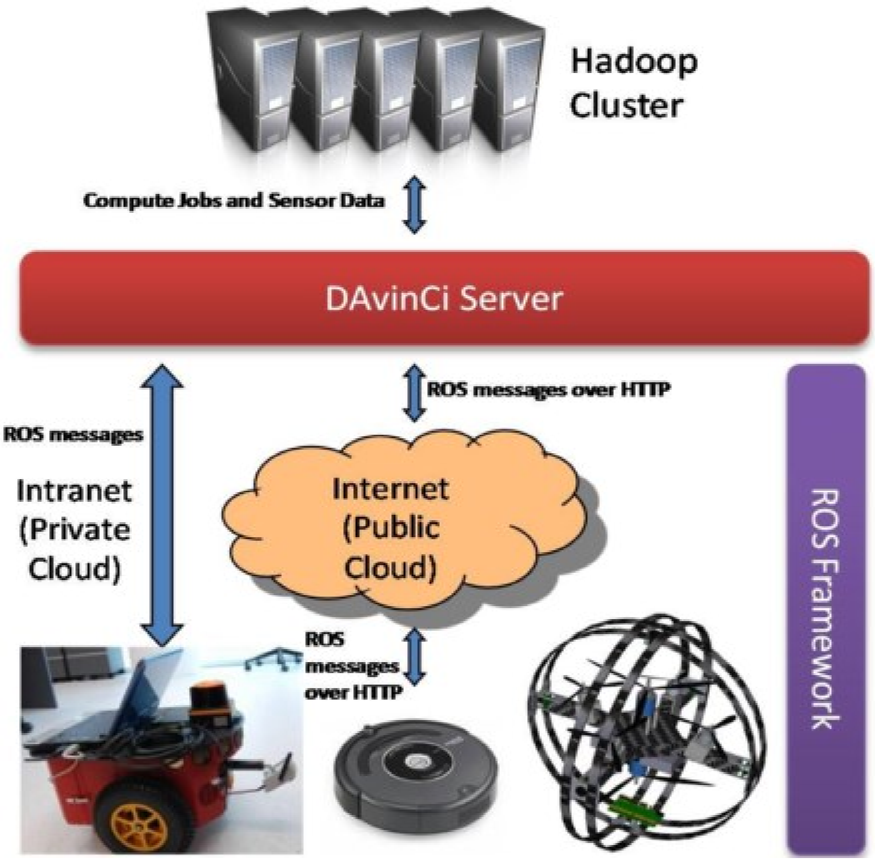

Figure 3-3: The DAvinCi architecture [24]

Figure 3-3 shows the high level overview of DAvinCi. The DAvinCi server acts as a service provider to robotic applications as well as a user of SaaS applications. For the development phase, DAvinCi architecture uses ROS, Hadoop file system and MapReduce algorithm. It collects data from robotic applications using the ROS. As described in the previous sub-section, ROS considers every process as a separate node. To collect data from robots, the DaVinCi server runs an ROS node which subscribes to all the ROS nodes running on these robots. Once the DaVinCi server collects all the data from the robots nodes, it pushes this data into the Hadoop file system. Hadoop is an open source software framework composed of the MapReduce algorithm and a distributed file system. MapReduce divides data processing into subtasks that can run in parallel in a distributed environment [25].

For the operation phase, after Hadoop receives the robots data, the DAvinCi server triggers MapReduce tasks to process this data. The DAvinCi server retrieves the processing results and sends them back to the robotic applications. The communication between DAvinCi server and robotic applications is done through ROS messages. ROS messages can be wrapped in HTTP requests for communication over the Internet.

Although DAvinCi allows robotic applications to request the execution of third party applications; it does not address the publication and discovery of third party applications nor their activation. Furthermore, it does not allow access to robotic applications anytime and anywhere for consumption purpose. Also, the interface technology used to enable third party applications is not web-based.

3.3.2.2.2 RoboEarth

RoboEarth [26] is a distributed learning database that allows robotic applications to generate, share and reuse knowledge. It is based on open source cloud architecture and contains knowledge that is encoded in Web Ontology Language (OWL) using Uniform Resource Identifiers (URIs) and typed links. In terms of execution, RoboEarth offers the possibility to re-use applications, as far as we consider knowledge that is shared by robots as applications.

RoboEarth's architecture is composed of three layers: a server layer, a generic components layer, and a skill abstraction layer. Figure 3-4 shows the RoboEarth three-layered architecture. The server layer contains the database and basic reasoning Web services. An example of data stored in the database could be a tea-cup; its Computer-Aided Design (CAD) model and the set of operations that can be done on the tea-cup, such as grab and hold. These operations are called action recipes. Action recipes are described using a high-level language.

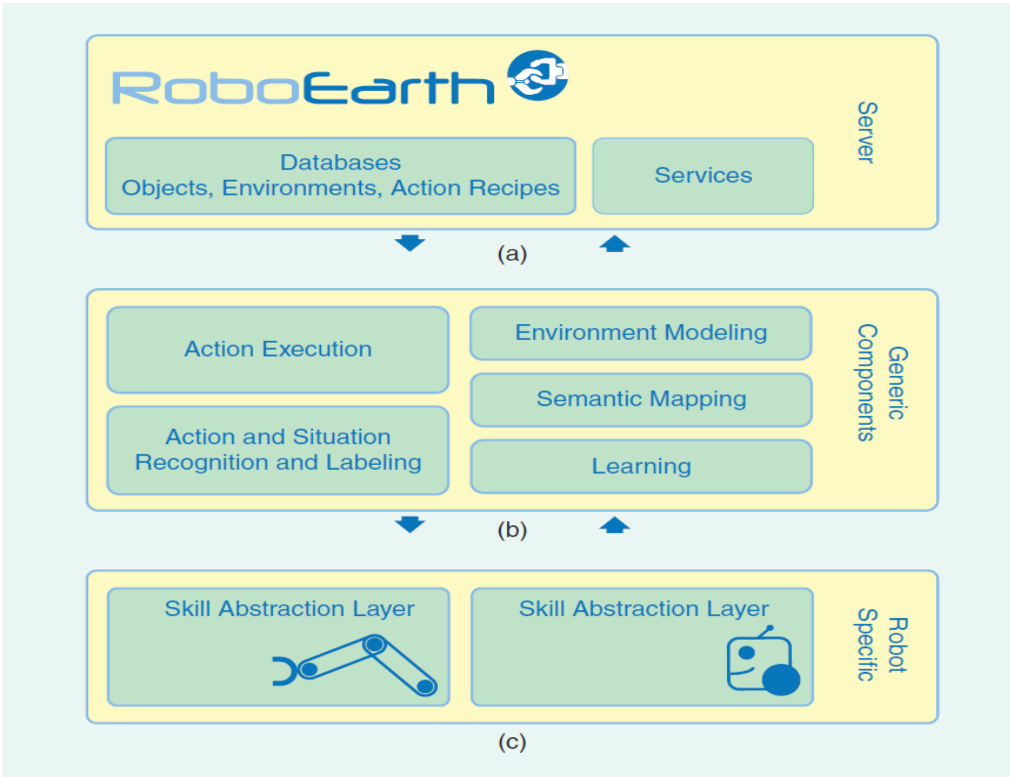

Figure 3-4: RoboEarth three-layered architecture [26]

In terms of operation, the reasoning service is responsible for mapping action recipes to robot skills to determine if a robot is capable of executing a given action recipe. RoboEarth's generic component layer allows robotic applications to interpret action recipes. The skill abstraction layer gives a generic interface to hardware-dependent robots capabilities.

For the interface technology, the server layer has three interfaces, which are:

- A Web interface that allows humans to label robot perceptions using Hyper Text Mark-up Language (HTML);
- A Representation State Transfer (REST) interface, utilized for communication between robotic applications and the knowledge database; and

- A publish/subscribe-based interface that allows robotic applications to subscribe to topics of interest while avoiding unnecessary communication traffic.

Although RoboEarth allows discovery and publication of applications to be executed by robotic applications, these applications are limited to knowledge acquired by other robots. Furthermore, these applications are not used as building blocks to create new robotic applications. Moreover, the application activation was not addressed. Access anytime and anywhere to robotic applications for consumption purpose is not addressed. The interface technology adopted in application re-use is web-based.

3.3.2.2.3 Senior Companion Robot System (SCRS)

SCRS [27] presents a cloud-based interactive system in which robots understand speech semantics and respond appropriately. The end-user interacts with the robot using voice commands or text. Figure 3-5 shows an example of human-robot interaction in the SCRS system. SCRS covers the execution phase of the application provisioning lifecycle. The system includes two types of third party applications. The first type of applications provides computation capabilities to robotic applications, like speech recognition and face identification. The robotic applications can re-use for operation purpose. The second type is monitoring applications that allow the end-user to remotely control and monitor the robot. The monitoring applications are offered as web services. The robot’s behavior model is composed of three steps: start and connect to the SCRS, send user input to the SCRS and execute commands received from the SCRS.

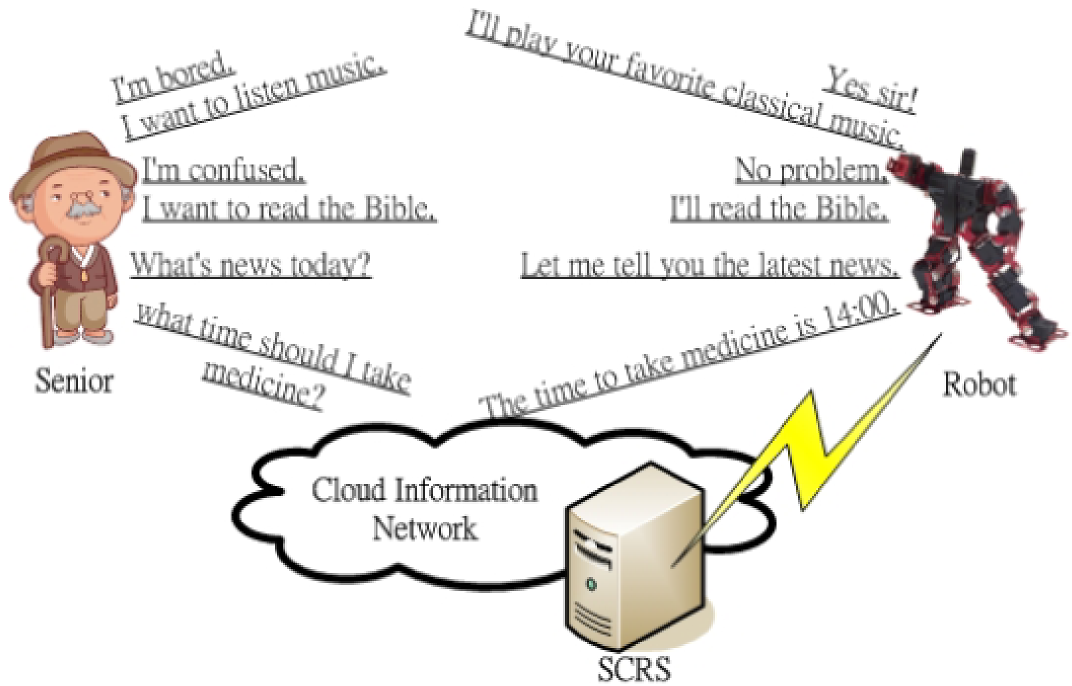

Figure 3-5: An example scenario of human-robot interaction in the SCRS system [27]

Even though SCRS allows robotic applications to request the execution of third party applications, it does not enable discovery and publication of third party applications that can be re-used as building blocks to create new robotic applications. Moreover, there is no application activation. The interface technology used to execute the third party applications is not mentioned, and access to robotic applications for operation purpose is not addressed.

3.3.2.2.4 Cloud-based approach to robot vision

Reference [28] presents a cloud-based approach to solving complex robot vision tasks using a smart camera system. Figure 3-6 shows the software architecture. This work covers the execution phase of the application provisioning lifecycle. The smart camera system is composed of a camera and a computation system that can be embedded or maintained in external dedicated hardware. It generates image information instead of raw image data. The smart camera system sends the image information generated by the camera to the robotic applications. The latter distribute the processing of image information on the computation systems surrounding the robot that are reachable via wireless network connection. The smart camera system is connected to the robot through a Gigabit Ethernet. The interface technology used for distributing image processing on the surrounding computation systems is Roblet framework.

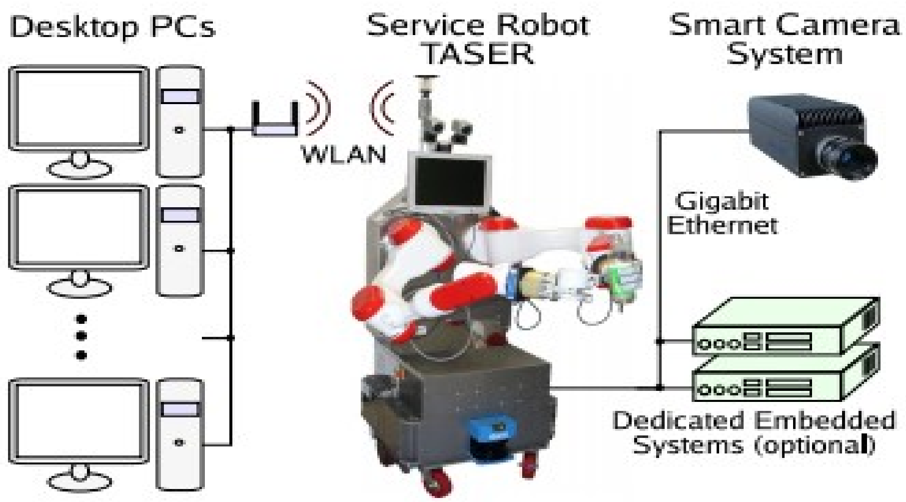

Figure 3-6: The software architecture for the distributed image processing [28]

This work allows robotic applications to request the execution of third party applications, but it is limited to image processing applications. The interface technology used to interact with the third party applications is tied to one technology, and is not web-based.

3.3.2.2.5 Simulation Environment for Cloud Computing Robots

Reference [29] presents a simulation environment for robotic applications using cloud services. It is concerned with the execution phase of the development lifecycle. It is based on the premise that the main obstacle for robotic applications to offer a real time response when using cloud services is the large number of routers between the robot and the cloud servers. It assumes that robots connect to clouds using a Wireless Access Network (WAN).

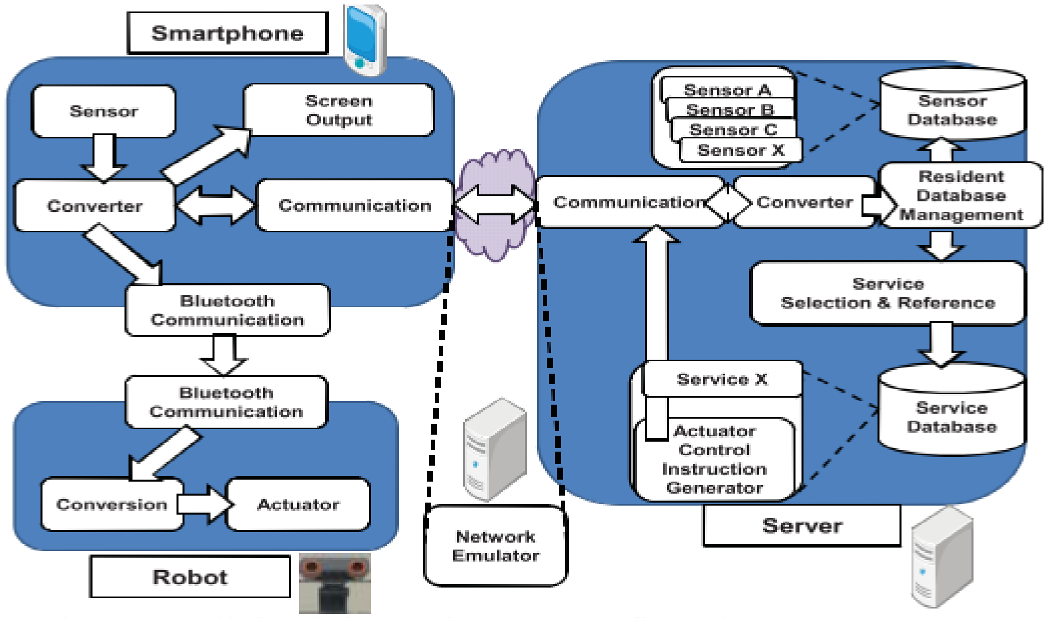

Figure 3-7: The simulation environment setup [29]

As shown in Figure 3-7, the simulation environment setup is composed of a Lego Mindstorm NXT robot, an android smart phone, a computer that plays the role of a server, and a network emulator that emulates wireless connection between the smart phone and the server. The simulation steps are as follows: the robot collects sensor data and sends it to the smart phone; the smart phone sends the collected data to the server, which processes the data and generates control instructions for the robot. The server sends the instructions back to the smart phone which sends them to the robot.

The first experiment consisted of cloning the smartphone application on a computer that was connected to the server through Ethernet. The computer played the role of the robot by sending sensor data to the server. The response time was calculated from the moment the computer sent sensing data until the moment it received actuation instructions from the server. Then, the number of computers running the smartphone application was increased to two, then three computers. The results showed that the response time increased by 40ms for each additional computer.

The second experiment calculated the same response time, but the smartphone application was running on the smartphone itself. The connection to the server was emulated and different delay times were yield. The response time was higher than the one achieved on the first experiment.

The results showed that an increase in the number of robots connected to the server generates an increase in the response time. Furthermore, using a smartphone to communicate with the server increases the response time.

This work offered the possibility for robotic applications to request the execution of third party applications (sensor data processing and actuation instruction generation), but it did not address the access anytime and anywhere for consumption purpose to robotic applications.

3.3.2.3 Inter-cloud Architectures for Application Provisioning

In this sub-section, we introduce work done on the inter-cloud architecture. First, we present the cloud broker framework which aims at decreasing customer costs when using cloud services pertaining to multiple clouds [31]. Then, we present the Next Generation Service Overlay Network (NGSON): a framework for inter-cloud operations with a focus on cloud interoperability [33].

3.3.2.3.1 Cloud Broker Framework

In [31], the authors present a cloud broker framework that allows users to request the execution of applications pertaining to different clouds, while decreasing the customer costs. The framework is based on the concept of brokers between users and cloud providers. The user makes graph-based requests to the cloud broker, where each node represents the desired resource and each link represents the required bandwidth and Quality of Service (QoS). The cloud framework broker is responsible for splitting the customers' requests, using an algorithm that aims at finding the optimal cloud resource allocation in terms of operation costs. Then, it executes the request by interacting with the cloud manager of each cloud.

Figure 3-8 shows the framework components. The authors use OpenFlow to establish and manage inter-cloud providers communication. OpenFlow is a network communication protocol that allows software applications to have direct access and control to the network devices [34].

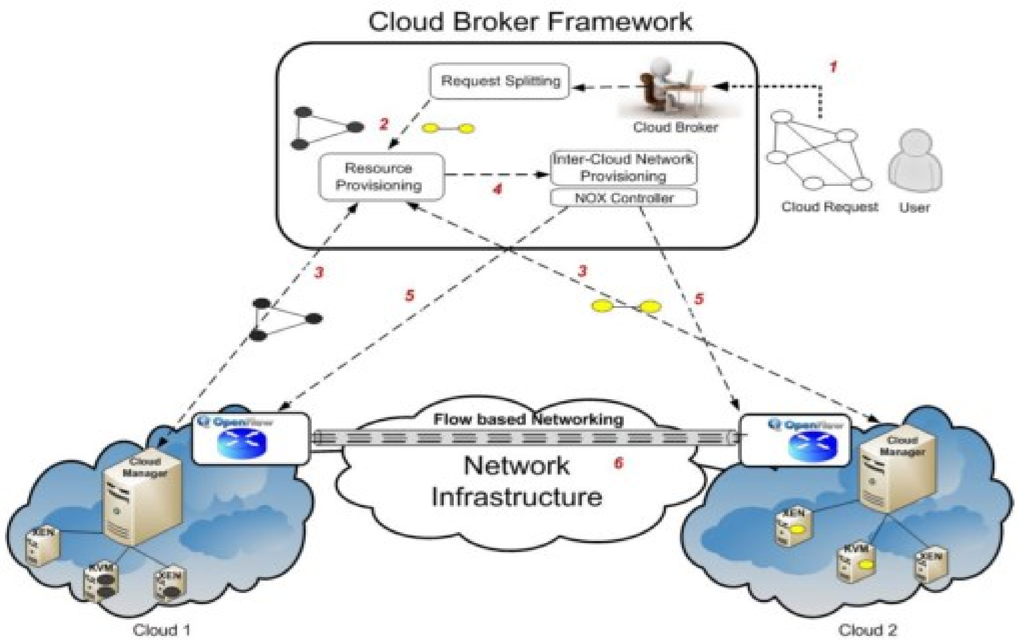

Figure 3-8: The interaction framework components [31]

This work offered the possibility for applications to request the execution of other applications, but it is limited to cloud applications that provide CPU, memory, and other cloud resources, and it does not allow the creation of new applications by using these cloud applications. The publication of the applications to the broker is not mentioned. Furthermore, the technology used to execute the applications requires making changes to existing clouds by adding new components (i.e., Openflow node).

3.3.2.3.2 NGSON

NGSON [33] provides a framework that allows end-users to discover and compose applications. Figure 3-9 introduces the main components of NGSON. The Identity management component offers a unified entry point to authenticate end-users of different cloud applications.

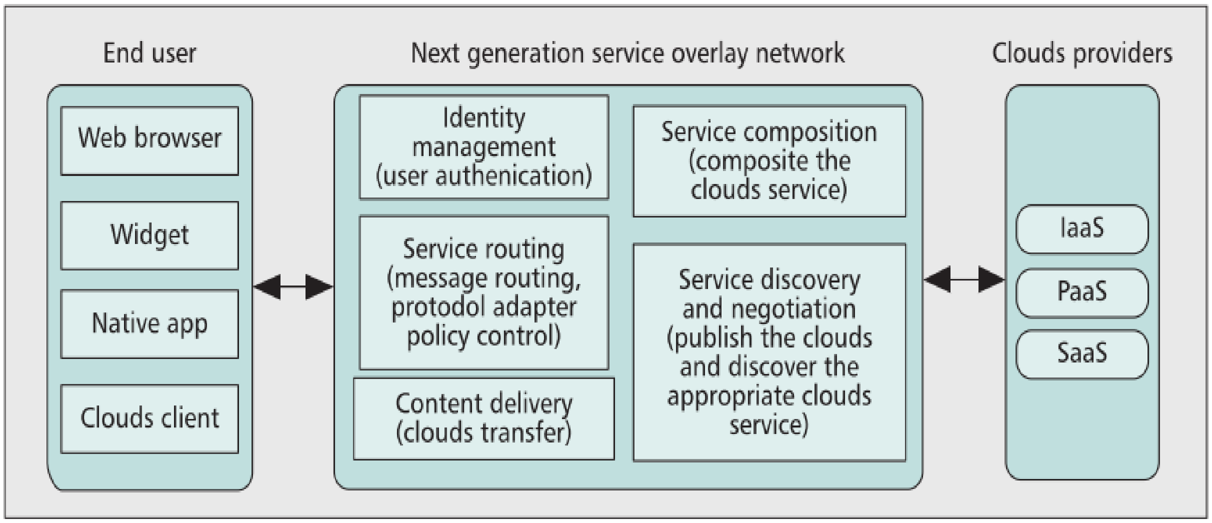

Figure 3-9: The main components of the NGSON [33]

The service composition component is responsible for the execution of composite services according to the service logic. The service routing component provides message routing, policy control and protocol adaptation for operations among different clouds. The content delivery component provides the possibility to deliver content from one service to another, from an end-user to another and between end-users and services. The service discovery component allows the services to publish their information on NGSON. It also supports the service discovery based on given criteria. Service routing in NGSON is based on IDs that are assigned to each service, called service ID. An instance ID is attached to each service for each cloud. When the end-user wants to start or stop a service instance, it sends a management request with the instance ID to NGSON. NGSON routes the request to the corresponding service in the corresponding cloud. The interface technology between the end-user and NGSON can be SOAP, REST or HTTP. NGSON offers protocol adaptation for inter-cloud operations. Although NGSON allows end-users to discover and activate cloud applications for operation purpose, the discovery mechanism is based on IDs that NGSON assigns to these applications and that the end-user must provide.

3.3.3 State of the Art Summary

Table 1 shows a summary of the state of the art review.

Table 1: The state of the art summary

State of the Art \ Requirements			General Requirement	Development Requirements		Deployment Requirement	Operation Requirements	
			Web-based interface technology	**Publication**	**Discovery**	**Activation**	**Execution**	**Access Anywhere Anytime**
Non-cloud Robotic Application Provisioning		17	Satisfied	Not Satisfied	Not Satisfied	Not Satisfied	Partially Satisfied	Partially Satisfied
		18	Not Satisfied	Partially Satisfied	Partially Satisfied	Not Addressed	Satisfied	Not Satisfied
Cloud-based Robotic Application Provisioning Architectures	Robotic apps as Service	21	Satisfied	Partially Satisfied	Partially Satisfied	Not Addressed	Satisfied	Not Addressed
		24	Satisfied	Partially Satisfied	Partially Satisfied	Not Addressed	Satisfied	Not Addressed
	Robotic apps as cloud users	24	Not Satisfied	Not Addressed	Not Addressed	Not Addressed	Partially Satisfied	Not Satisfied
		26	Satisfied	Partially Satisfied	Partially Satisfied	Not Addressed	Partially Satisfied	Not Addressed
		27	Not Addressed	Not Satisfied	Not Satisfied	Not Satisfied	Satisfied	Not Addressed
		28	Not Satisfied	Not Addressed	Not Addressed	Not Addressed	Satisfied	Not Addressed
		29	Satisfied	Not Addressed	Not Addressed	Not Addressed	Satisfied	Not Addressed
	Inter-cloud	31	Not Satisfied	Not Addressed	Not Addressed	Not Addressed	Partially Satisfied	Not Addressed
		33	Satisfied	Partially satisfied	Partially Satisfied	Partially Satisfied	Partially Satisfied	Satisfied

3.4 Chapter Summary

In this chapter we presented the wildfire suppression application domain that illustrates the need of a cloud-based architecture for robotic application.

Then, we used the application provisioning lifecycle to derive requirements from the application domain. We categorized these requirements into four groups: general, development-related, deployment-related, and operation-related requirements.

Finally, we presented the state of the art based on the identified requirements. We categorized the state of the art into two groups: non-cloud robotic application provisioning solutions and cloud-based robotic application provisioning architectures. We evaluated the state of the art using the requirements we previously identified. We concluded that none of the works presented in the state of the art meet all the requirements we identified.

4 A Business Model for Cloud-Based Robotic Application Provisioning

Business models are tools that describe business logics using a set of object, concepts and their relationship [16]. Business models are used in information systems to help increase the mutual understanding between the business and the information system domains; they create shared understanding and a common language [16]. They are used as starting point for architecture design. They do not define the actual technologies that are used.

In this chapter, we propose a business model that tackles the issue of robotic application and third party application cloud interaction for the purpose of cost efficiency in robotic application provisioning. The business model can be used as the starting point for cloud-based architectures for robotic application provisioning. We start by presenting the assumptions we used in designing the business model. Then, we define the actors. Next, we present the interactions between these actors.

4.1 Assumptions

The first assumption is that all the robotic applications as well as the third party applications we considered are offered as cloud services. This will allow access to robotic applications for consumption purpose anytime and anywhere. Furthermore, it will also allow third party applications to be accessed for discovery, activation, and operation anytime, anywhere.

4.2 Actors

The business model we propose introduces actors categorized in four groups: the end-user, robotic cloud providers, third party cloud providers, and the interaction framework provider. Each of these actors may play one or several roles in the business model. As an example, a robotic PaaS provider may also play the role of robotic IaaS provider. The technologies and architectural realization of these roles are defined in the chapter on the architecture. We focus on the interaction provider role. Figure 4-1 shows the actors, the business roles, and the interactions.

The End-User

The role of the end-user is to discover and consume robotic applications.

The Robotic Cloud Providers

The robotic cloud providers are responsible for creating, managing, and providing robotic applications. This group comprises the robotic SaaS provider, the robotic PaaS provider, and the robotic IaaS provider. The robotic SaaS provider offers robotic applications to end-users for discovery and consumption. The robotic PaaS provider enables the creation and deployment of robotic applications by offering the necessary development frameworks, libraries, and tools. It also enables the creation of new robotic applications re-using third party applications. Hence the robotic PaaS provider discovers, activates and executes third party applications. The robotic IaaS provider is responsible for operating the robotic applications.

The Third Party Cloud Providers

The third party cloud providers offer third party applications that can be re-used as building blocks to create new applications. This group contains the third party SaaS provider, the third party PaaS provider, and the third party IaaS provider. The third party SaaS provider publishes its applications to the interaction framework provider. Third party applications can be primitive applications that are typically used by other applications or full-fledged applications that support end-users. The role of the third party PaaS provider is to develop, deploy, and operate the third party applications. The third party IaaS is responsible for deploying and operating the third party application.

The interaction Framework Provider

The role of the interaction framework provider is to offer a platform that enables the interactions between the robotic platform provider and the third party application providers.

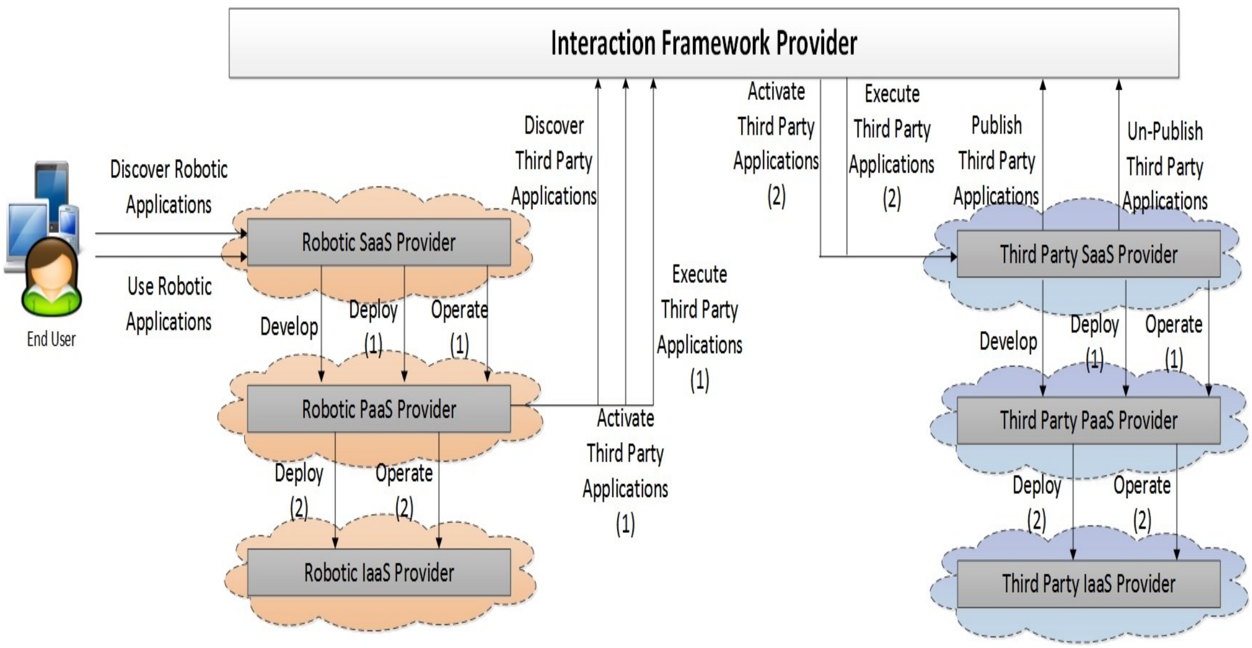

Figure 4-1: The business roles and interactions

4.3 Interactions of the Business Model

The following sub-section describes the interactions between the different actors in the business model. The technologies and the methods used to carry out the interaction framework provider interactions will be defined in the architecture. The other interactions are out of the scope of this thesis.

The interactions can be categorized in four groups: end-user, robotic cloud providers, third party cloud providers, and framework interaction provider interactions. The interactions in the robotic cloud providers group describe the same operations as in the third party cloud providers group. For the sake of simplicity we will describe only the robotic cloud providers' interactions.

4.3.1 End-User Interactions

This group contains interactions between the end-user and the robotic SaaS provider.

The Discover Robotic Application Interaction: This interaction allows the end-user to find the robotic application that suits its need. The discovery may involve a broker that plays the role of initial intermediate between the end-user and the robotic application provider.

The Use Robotic Application Interaction: This interaction is concerned with the operation of the robotic application by the end-user. The use of the robotic application involves usage monitoring, billing, and SLA monitoring.

4.3.2 The Robotic Cloud Provider Interactions

This group contains the interactions between the robotic SaaS, the robotic PaaS, and the robotic IaaS providers.

Develop Robotic Application Interaction: The robotic SaaS provider is responsible for developing the applications that the robotic SaaS provider offers to end-users. This interaction involves the analysis, design, implementation, and testing of the robotic application.

Deploy Robotic Application (1) Interaction: This interaction happens after the development interaction. Deployment of the robotic application involves installation and activation.

Operate Robotic Application (1) Interaction: This interaction describes the execution, the utilization and the run-time management of the robotic application.

Deploy Robotic Application (2) Interaction: This interaction describes the introduction of the robotic application in the infrastructure. It takes into account the requirements and guidelines on how to deploy the application in a physical system.

Operate Robotic Application (2) Interaction: This interaction is meant to hide the heterogeneity and the distribution of the underlying resources from the robotic application PaaS provider.

4.3.3 Interaction Framework Provider Interactions

This group is categorized into two sub-groups: interactions with the robotic cloud providers and interactions with the third party cloud providers.

4.3.3.1 Robotic Cloud Providers Interactions

Discover Third Party Application (1) Interaction: This interaction allows robotic PaaS providers to find the third party applications that are offered by the third party SaaS providers. The interaction framework provider uses the applications that have been previously published to respond to discovery requests. It is responsible for identifying and locating the appropriate third party applications that match to the robotic PaaS provider request. The architecture that will derive from the business model will define the methods used to discover the applications.

Activate Third Party Application (1) Interaction: This interaction is used to initiate the third party application use. Before the robotic applications can re-use third party applications, the third party applications need to be activated.

Execute Third Party Application (1) Interaction: This interaction represents the actual use of the third party application. After the robotic PaaS cloud provider activates the third party application that it wants to re-use, it can start executing it.

4.3.3.2 Third Party Cloud Providers Interactions

Publish Third Party Application Interaction: Through this interaction, the third party SaaS provider describes applications to the interaction framework provider.

Activate Third Party Application (2) Interaction: This interaction describes the same goal as the activate third party application (1) interaction, except that it occurs between the interaction framework provider and the third party SaaS provider.

Execute Third Party Application (2) Interaction: This interaction describes the same goal as the execute third party application (1) interaction, but it is between the interaction framework provider and the third party SaaS provider.

Un-Publish Third Party Application Interaction: The third party SaaS providers may choose to stop making some of its applications available in the interaction framework. The un-publish interaction is used to allow the third party SaaS providers to withdraw their applications.

4.4 The Proposed Business Model Applied to the Wildfire Suppression Scenario

Let us illustrate the business model roles and interactions using the wildfire suppression scenario. The actors are: the wildfire suppression robotic provider, the robotic PaaS provider, the robotic IaaS provider, the supply management provider, the fire detection provider, the third party PaaS provider, the third party IaaS provider, the interaction framework provider, and the end-user.

Figure 4-2 shows the sequence diagram of the end-to-end execution of the scenario. For the sake of simplicity, we will consider that the fire detection and the supply management providers use the same third party PaaS provider and the same third party IaaS provider.

To offer the fire detection application, the fire detection provider must request the development of the application from the third party PaaS provider. The third party PaaS provider and the third party IaaS provider are involved in the development, deployment, and the operation of the fire detection and the supply management applications. The fire detection and the supply management' providers publish their applications in the interaction framework.

To develop the wildfire suppression application, the robotic PaaS provider discovers the two third party applications: fire detection and supply management. The robotic PaaS provider composes the two applications to develop the wildfire suppression application. When the robotic PaaS provider deploys the wildfire suppression application, it also requests the activation of the fire detection and the supply management applications from the interaction framework provider.

When the end-user discovers and uses the wildfire suppression application, the wildfire suppression provider requests the execution of the fire detection and the supply management applications from the interaction framework provider. Once a fire occurs, the fire detection application notifies the wildfire suppression application. The latter requests the supply management application for the most adequate supply station for the robots to supply in fire retardants depending on criteria, such as type of retardant and ground or aerial accessibility.

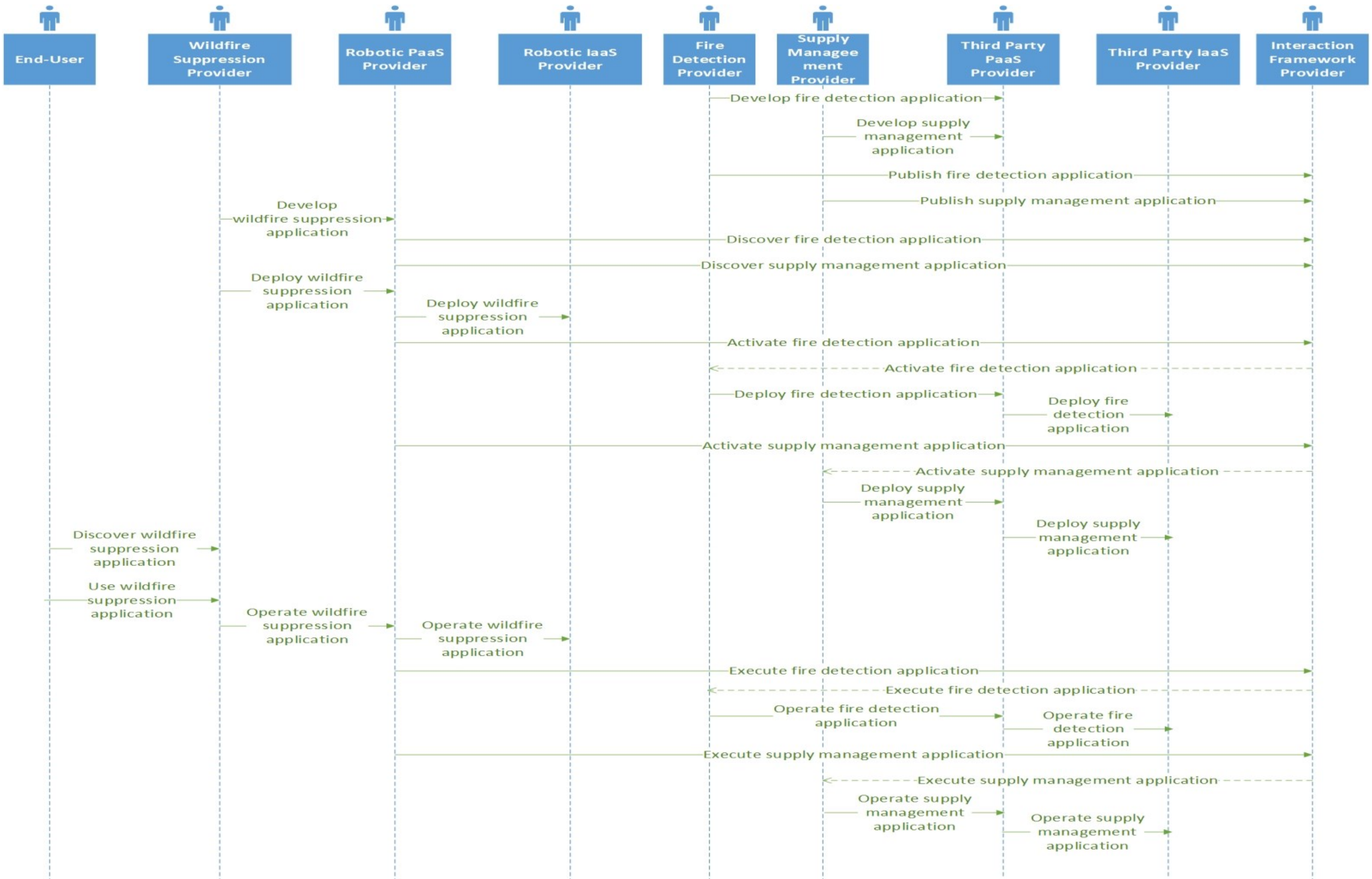

Figure 4-2: The sequence diagram of the end-to-end execution of the wildfire suppression scenario

4.5 Chapter Summary

In this chapter, we introduced the proposed business model with the assumptions we used in our design. We presented the business actors: the end-user, the robotic IaaS provider, the third party IaaS provider, the robotic PaaS provider, the third party PaaS provider, the robotic SaaS provider, the third party SaaS provider, and the interaction framework provider. Next, we introduced the interactions of the business model. Finally, we illustrated the business model using the wildfire suppression scenario. The rest of the thesis focuses on the interaction framework provider and its interactions.

5 Proposed Architecture

In this chapter we present the architecture of the interaction framework provider role described in the business model from the previous chapter. We start with an overall view of the architecture. Next, we describe the protocols and the nodes. Then, we present the procedures. Finally, we present the wildfire suppression use case

5.1 Overall View of the Architecture

Figure 5-1 shows an overview of the architecture. We use P2P overlay networks as the cornerstone of the proposed architecture because of the advantages they offer. P2P overlays allow easy sharing and discovery of information. They are self-organized networks, as they do not require central management of joining and leaving peers.

We chose P2P overlay networks over SOA paradigm for many reasons. Overlay networks are self-organized. Also, they do not require centralized management. Furthermore, overlay networks offer built-in scalability in terms of the number of nodes in the network. Moreover, they enable operations between entities while minimizing the number of interfaces.

In this section, we start by presenting the architectural principles we adopted in designing the architecture. Then, we give a high level description of the architecture.

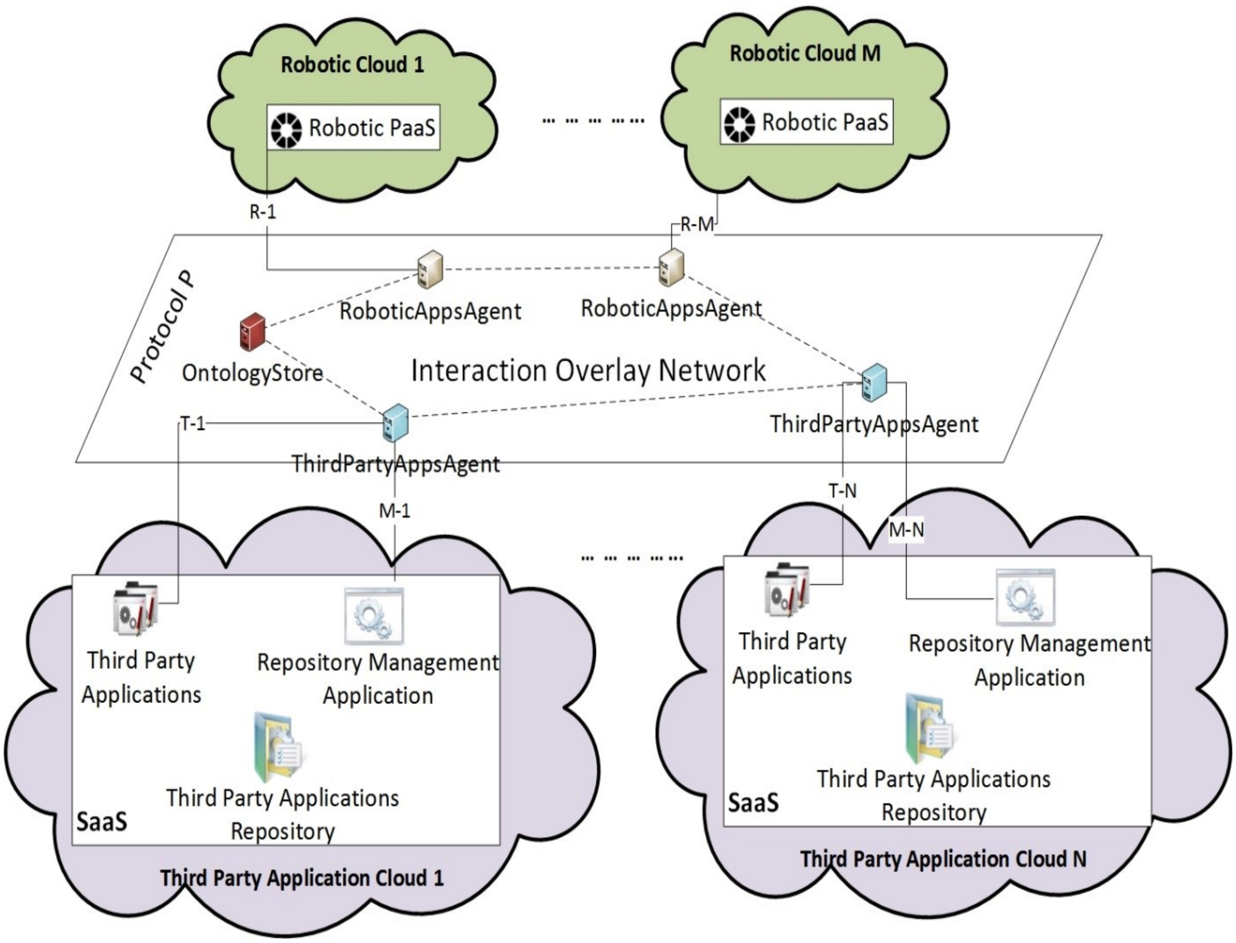

Figure 5-1: The interaction overlay architecture

5.1.1 Architectural Principles

To design the architecture, we followed a set of principles. The first principle is that each third party SaaS provider owns an application repository. This application repository has a repository management application that enables interactions between the application repository and the overlay. Application repository allows third party SaaS providers to store the information related to all the third party applications it offers to end-users and eventually other applications. It is common to use application repositories for application publication and discovery.

The second principle is that all third party SaaS providers use a common ontology to describe their applications. We use ontologies because they offer the possibility to describe the applications with a high-level of abstraction and in a refined fashion. The word "ontology" is used to describe the study of beings and the study of various kinds of beings [30]. In computer science, ontology refers to clearly defined, machine-processable specifications of shared abstract models [30].

This common ontology is stored in a node we call *OntologyStore*. It is offered and managed by the interaction framework provider. The robotic PaaS providers and the third party SaaS providers request a copy of the ontology when they join the Interaction Overlay Network. We assume that the third party SaaS providers use it to describe their applications off-line.

The third principle is that every robotic PaaS provider and every third party SaaS provider are represented by one and only one node in the overlay provided by the interaction framework provider. We chose to assign a node for each provider to allow providers full control over the interactions they have with the overlay. Also each provider has its own node for efficiency purpose; if a node fails it affects only that provider. The interactions between the robotic PaaS and the third party applications are done via the nodes that represent their respective providers in the overlay.

5.1.2 High-Level Description of the Architecture

We introduced in the previous chapter the interactions between the interaction framework provider and the other actors in the business model. The proposed architecture contains elements corresponding to the actors involved in these interactions. As shown in Figure 5-1, we have three types of nodes in the overlay: *RoboticAppsAgent, ThirdPartyAppsAgent,* and *OntologyStore.* A *RoboticAppsAgent* node represents a robotic PaaS provider, while a *ThirdPartyAppsAgent* node represents a third party application provider. The *OntologyStore* node does not correspond to any the cloud providers. We describe these nodes' types and their architecture in the up-coming sections.

R-1 to R-M are the interfaces between the robotic platforms and their corresponding *RoboticAppsAgent* nodes. T-1 to T-M are the interfaces between the third party application and their corresponding *ThirdPartyAppsAgent* nodes. M-1 to M-N are the interfaces between the third party repository management applications and their corresponding *ThirdPartyAppsAgent* nodes. Protocol P enables the interaction between the overlay nodes.

5.2 Detailed Architecture

We defined a request/reply protocol for the interactions between the overlay nodes. The protocol is presented in the first sub-section. Then we describe the nodes architecture.

5.2.1 Overlay Protocols

We defined two protocols: P1 is request/response protocol, P2 is subscribe/notify protocol. Protocol P1 uses the message summarized in Table 2. The "discover" message is a multicast since it is sent to all *ThirdPartyAppsAgents* nodes in the overlay to find the information of the desired third party application. When a *ThirdPartyAppsAgent* node receives a "discover" message it sends a response back to the *RoboticAppsAgent* node that issued the message. If it has a third party application that fits the criteria in the "discover" message, it sends the information about the third party application in the response. Otherwise, it sends an empty response.

The "activate" message is a unicast, because it is sent from to a specific *ThirdPartyAppsAgent* node to activate the third party application that this node represents. When a *ThirdPartyAppsAgent* node receives an "activate" message, it requests the activation of the desired third party application. The *ThirdPartyAppsAgent* node sends back the response that it receives from the third party application as a response to the "activate" message.

The “execute” message is a unicast, because it is sent to a specific *ThirdPartyAppsAgent* node to execute the third party application that this node represents. When a *ThirdPartyAppsAgent* node receives an “execute” message, it request the execution of the desired third party application. The *ThirdPartyAppsAgent* node sends back the response that it receives from the third party application as a response to the “execute” message.

Table 2: The overlay messages

Message type	Message description	Message address
Discover	Discovers the third party application requested by the robotic platform. Sent by the *RoboticAppsAgent* to the overlay network.	Multicast
Activate	Activates the specific third party application requested by the robotic platform. Sent by the *RoboticAppsAgent* to the *ThirdPartyAppsAgent*.	Unicast
Execute	Executes the specific third party application requested by the robotic platform. Sent by the *RoboticAppsAgent* to the *ThirdPartyAppsAgent*.	Unicast
Copy ontology	Requests a copy of the ontology from the *OntologyStore*. Sent by the *RoboticAppsAgent* and the *ThirdPartyAppsAgent*.	Unicast

Protocol 2 uses only the “execute” message. When a *ThirdPartyAppsAgent* node receives an “execute” message, it subscribes the *RoboticAppsAgent* node that sent the message. It also sends a subscription request to the third party application. When the *ThirdPartyAppsAgent* node receives a notification from the third party application, it sends it to the subscribed *RoboticAppsAgent* node. An illustrative example of this protocol use is the fire notification sent from the fire application to the wildfire suppression application in the wildfire suppression application domain.

5.2.2 Nodes Architecture

The three nodes' types have three common layers: the communication layer, the service layer, and the abstraction layer. Each layer has a set of functional components, some of which are common to all nodes' types, and some are specific to each node type. In the following sub-section, we describe the three layers. Then, we describe the functional components.

5.2.2.1 Nodes Layers

The communication layer enables nodes to communicate with other nodes in the overlay, the third party application repositories, the third party applications, and the robotic PaaS. It forwards the messages it receives to the service layer.

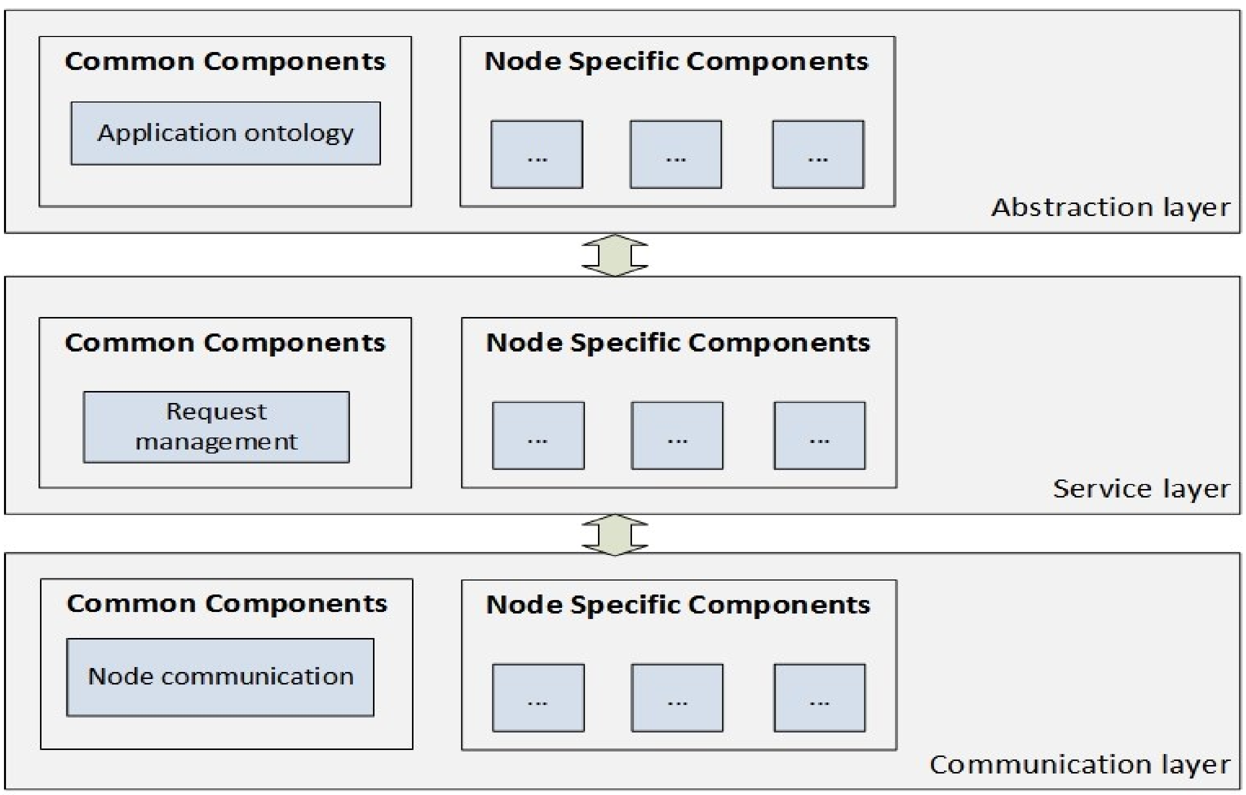

Figure 5-2: The view of the three nodes components

The service layer manages the requests in the overlay (i.e., publication, discovery, activation, execution, and ontology requests). It contains the operations corresponding to each request. The service layer uses the abstraction layer to respond to requests. The abstraction layer builds and maintains an abstract representation of the third party applications published in the overlay. As shown in Figure 5-2 the three layers contain components that are common to the three nodes, and components that are specific to each node.

5.2.2.2 Components in the Layers

In this sub-section, we describe the components that are common to every node in the overlay. Then, we describe the components that are node specific.

5.2.2.2.1 Common Components

Request Management

The request management sorts and interprets the requests that each node receives; these requests differ depending on the node type. The request management forwards the publication requests to the application publication component, the discovery requests to the application discovery component, the application activation requests to the activation component, and the execution requests to the application execution components. For the ontology requests, the request management gets a copy from the application ontology and sends it to the node issuing the request through the node communication component.

Node Communication

The node communication allows communication with other nodes in the overlay. It receives the requests coming from other nodes in the overlay. It also sends the request coming from the discovery, activation, and execution components to the other nodes in the overlay.

Application Ontology

The application ontology component holds a copy of the common ontology that is used to describe the third party applications that are offered by the third party SaaS providers that join the

Interaction Overlay Network. The request management component uses the application ontology component to interpret requests (i.e., publication, discovery, activation, execution, ontology requests).

5.2.2.2.2 Node-Specific Components

Figures 5-3 and 5-4 show the node-specific components present in the *ThirdPartyAppsAgent*, the *RoboticAppsAgent* respectively. The *OntologyStore* node architecture contains only the common components.

Application Activation

The application activation component is responsible for requesting the activation of the requested third party application. It sends the activation request to the third party application through the application communication component.

Application Execution

The application execution component receives third party application execution requests from the robotic platform. It sends the request to the appropriate *ThirdPartyAppsAgent* node that represents the third party application requested for execution. When it receives the execution response from the third party application, it sends it back to the robotic platform.

Application Publication

The application publication receives publication requests from the third party application repository management application. It updates the categorized application repository.

Platform Communication

The platform communication component is responsible for the interactions with the robotic platform.

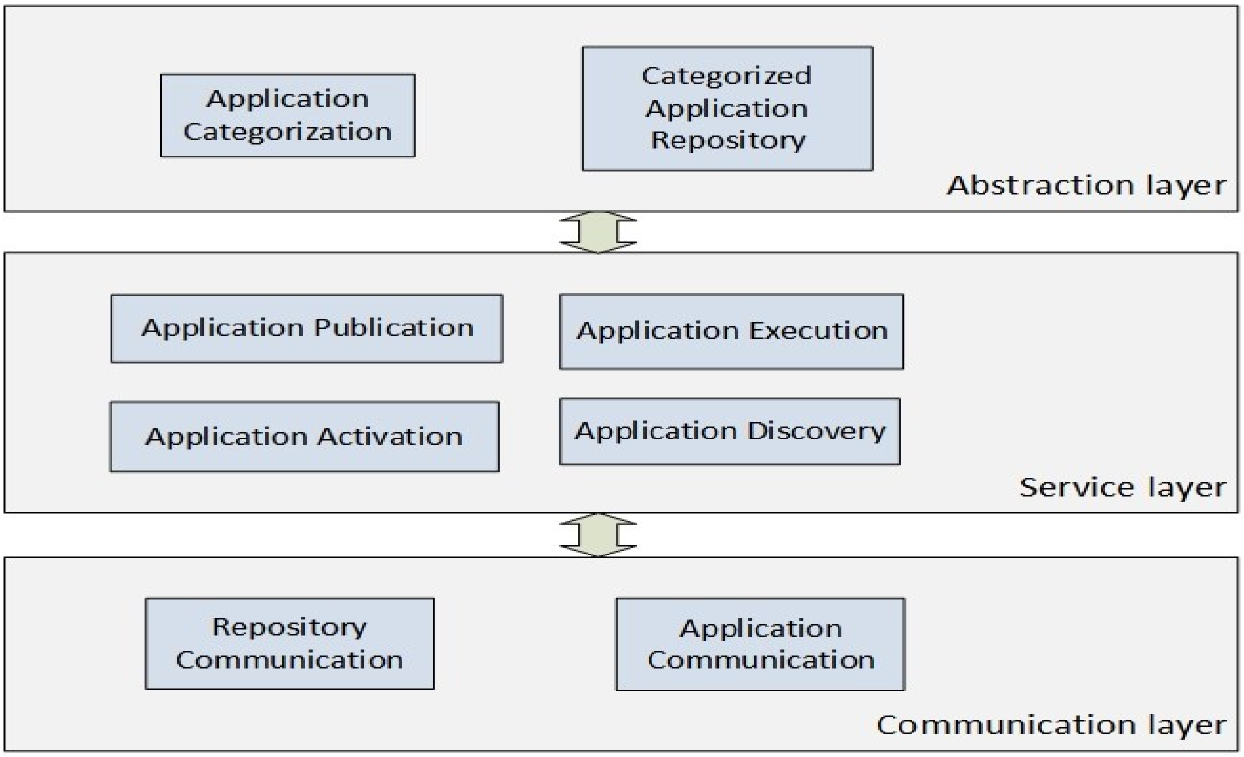

Figure 5-3: The node-specific components of the ThirdPartyAppsAgent node

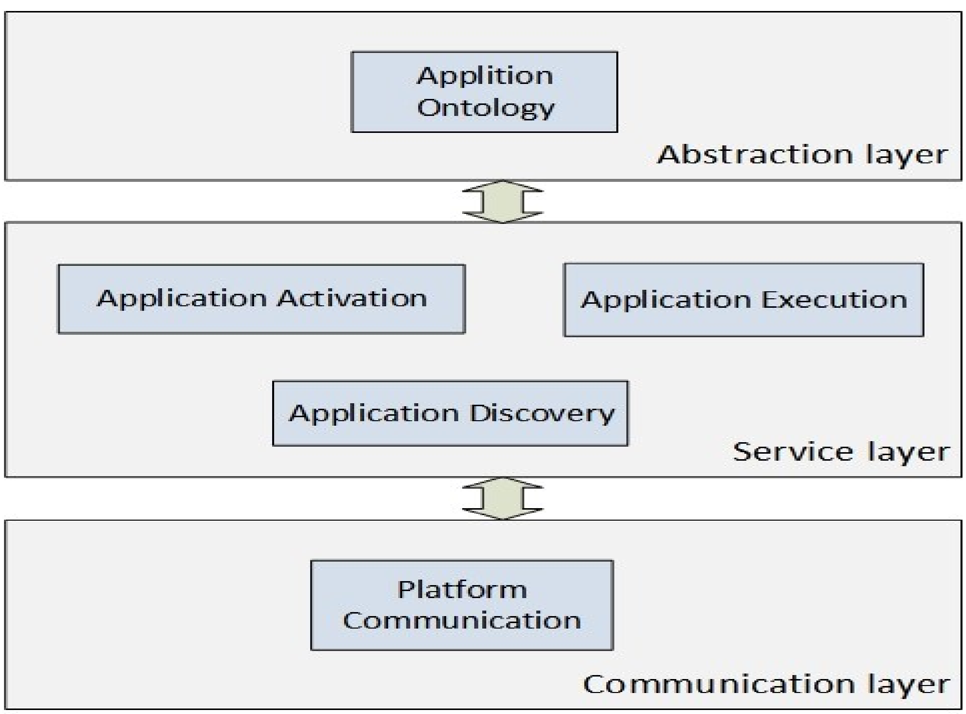

Figure 5-4: The node-specific components of the RoboticAppsAgent node

Application Discovery

In the *RoboticAppsAgent* node, the application discovery component receives third party application discovery requests from the robotic platform. It sends the discovery request to the *ThirdPartyAppsAgent* node in the overlay.

In the *ThirdPartyAppsAgent* node, it looks up the third party application requested for discovery in the categorized application repository. It sends back the discovery response to the *RoboticAppsAgent* node.

Application Categorization

When the application categorization receives a publication request from the publication application component, it updates the categorized application repository with the application information.

Categorized Application Repository

The categorized application repository stores a description of the third party applications that are published in the overlay. It contains the operations that the third party application exposes. It also contains information about the third party application routing.

Repository Communication

The repository communication component is responsible for the interactions with the third party applications repository that pertain to the cloud that the *ThirdPartyAppsAgent* represents.

Application Communication

The application communication component is responsible for the interactions with the third party applications.

5.3 Overlay Procedures

The overlay procedures include the management procedures and functional procedures.

5.3.1 Management Procedures

For the management procedures, we can identify the following procedures: joining and voluntary departure. The *RoboticAppsAgent* and *ThirdPartyAppsAgent* nodes join the overlay on behalf of the providers they represent.

5.3.1.1 Joining Procedure

Both the *RoboticAppsAgent* and the *ThirdPartyAppsAgent* nodes need to join the overlay network. We start by explaining the *ThirdPartyAppsAgent* node joining procedure, then the *RoboticAppsAgent* procedure. The Interaction Overlay Network provider manages the creation of the *OntologyStore* node. It is the first node in the overlay network; hence it doesn't join the network per se.

We assume that the repository management application triggers the *ThirdPartyAppsAgent* node joining procedure. To join the overlay network, the *ThirdPartyAppsAgent* requests the ontology from the *OntologyStore* node. Figure 5-5 shows the sequence diagram of the joining procedure for the *ThirdPartyAppsAgent*. The *ThirdPartyAppsAgent* sends a request to the *OntologyStore*. The *OntologyStore* interprets the request. Then, it sends the response back to the *ThirdPartyAppsAgent*. Afterward, the *ThirdPartyAppsAgent* stores the ontology copy locally in its abstraction layer. The ontology is later used to interpret requests, and the information in the categorized application repository.

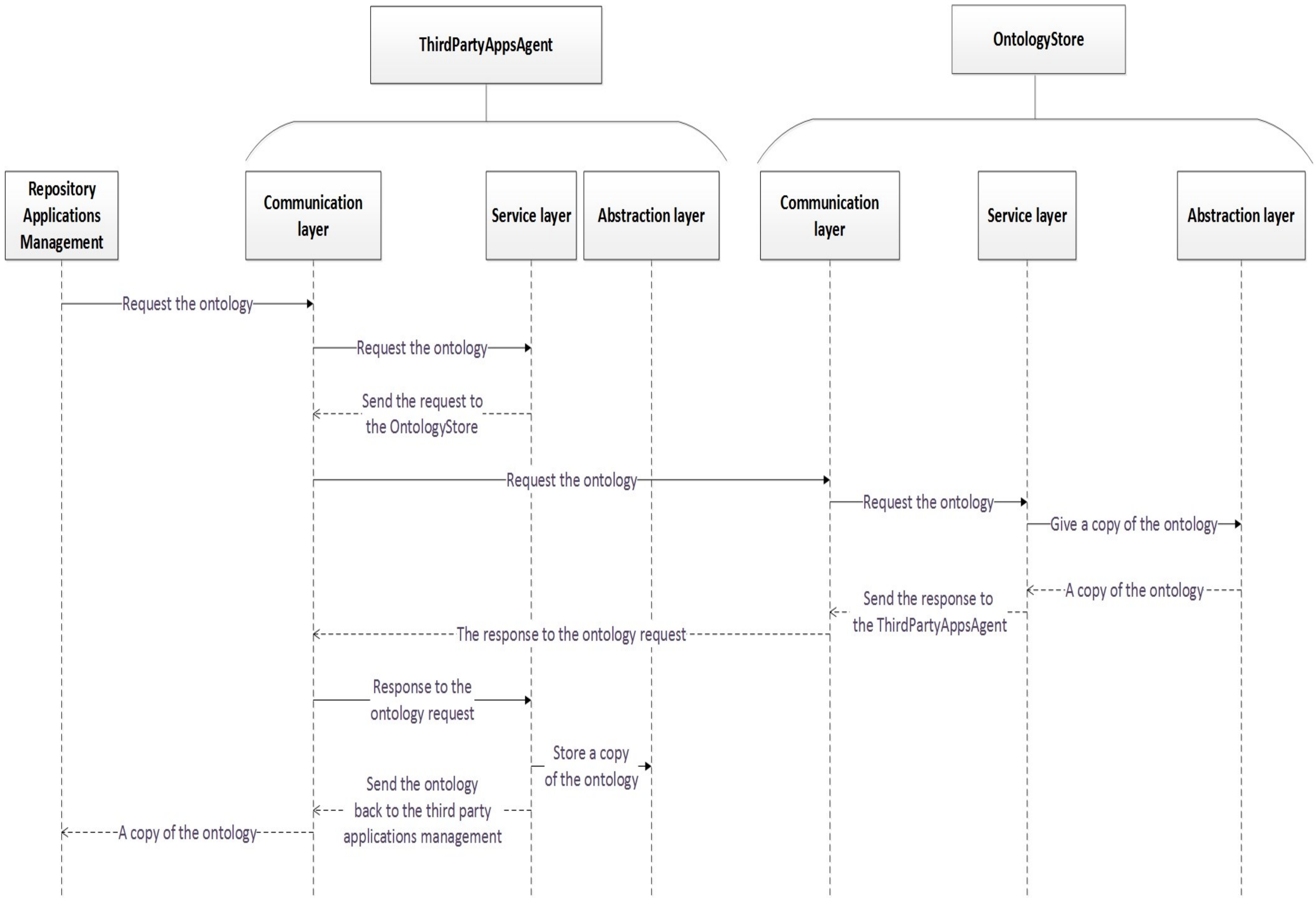

Figure 5-5: The sequence diagram of the joining procedure of the ThirdPartyAppsAgent node

We assume that the robotic platform triggers the *RoboticAppsAgent* joining procedure through a management application. To join the overlay network, the *RoboticAppsAgent* requests the ontology from the *OntologyStore*. Figure 5-6 shows a sequence diagram of the *RoboticAppsAgent* joining procedure. The *RoboticAppsAgent* sends a request to the *OntologyStore*. Once the *OntologyStore* receives the request, it sends the ontology back to the *RoboticAppsAgent*. The *RoboticAppsAgent* stores a copy of the ontology in its local abstraction layer. The ontology is later used to interpret requests.

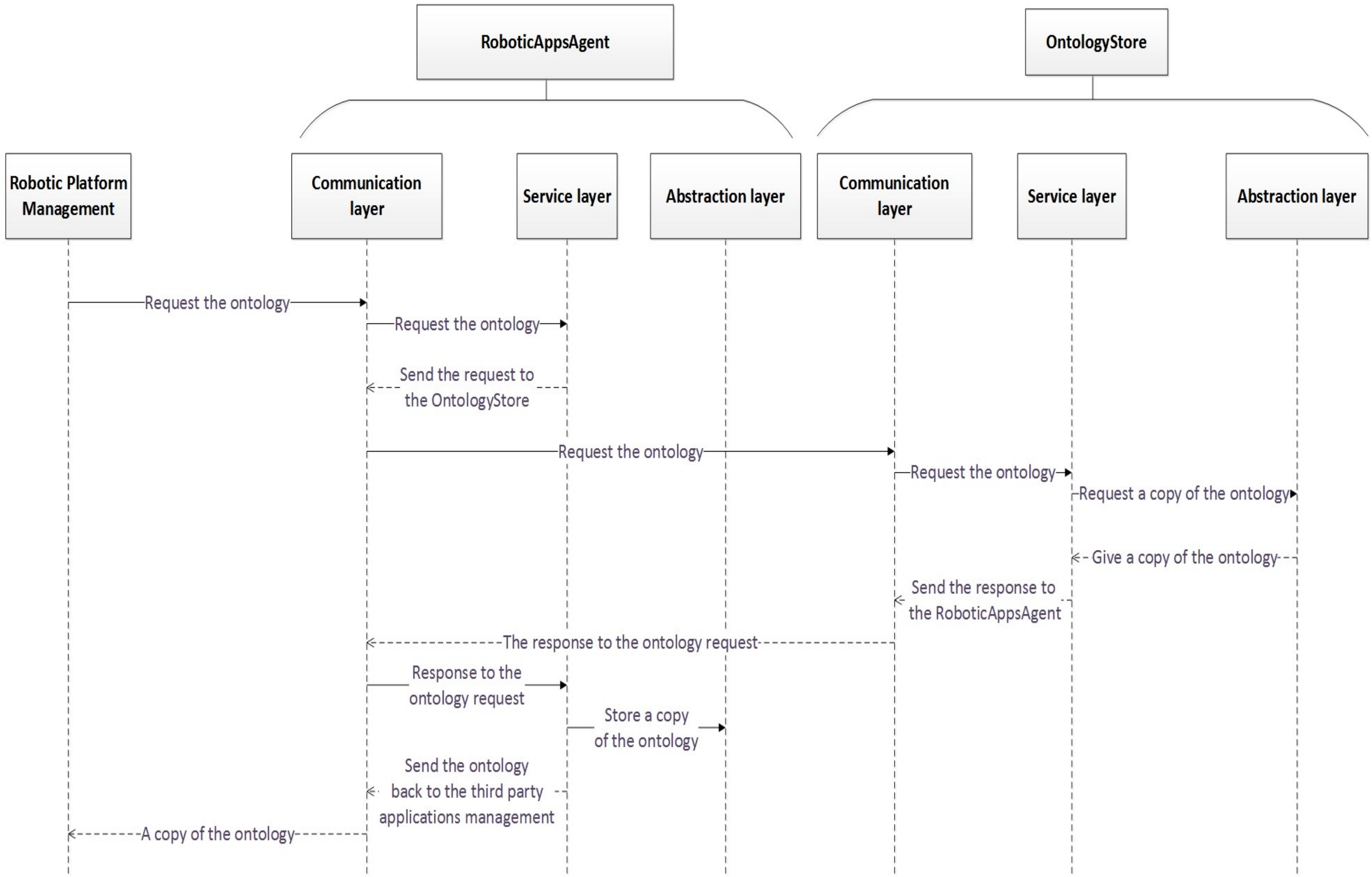

Figure 5-6: The sequence diagram of the joining procedure of the RoboticAppsAgent node

5.3.1.2 Departure Procedure

Both the *RoboticAppsAgent* and the *ThirdPartyAppsAgent* nodes may voluntarily leave the overlay network at any time. The leaving procedure of the *ThirdPartyAppsAgent* node requires notifying the other nodes in the overlay, while the leaving procedure of the *RoboticAppsAgent* node does not require any action in the overlay. Therefore, we discuss only the *ThirdPartyAppsAgent* node departure procedure.

We assume that the third repository management application triggers the departure procedure of *ThirdPartyAppsAgent* node in the overlay. Figure 5-7 shows the leaving procedure of the *ThirdPartyAppsAgent*. To leave the overlay network, The *ThirdPartyAppsAgent* empties its local abstraction layer.

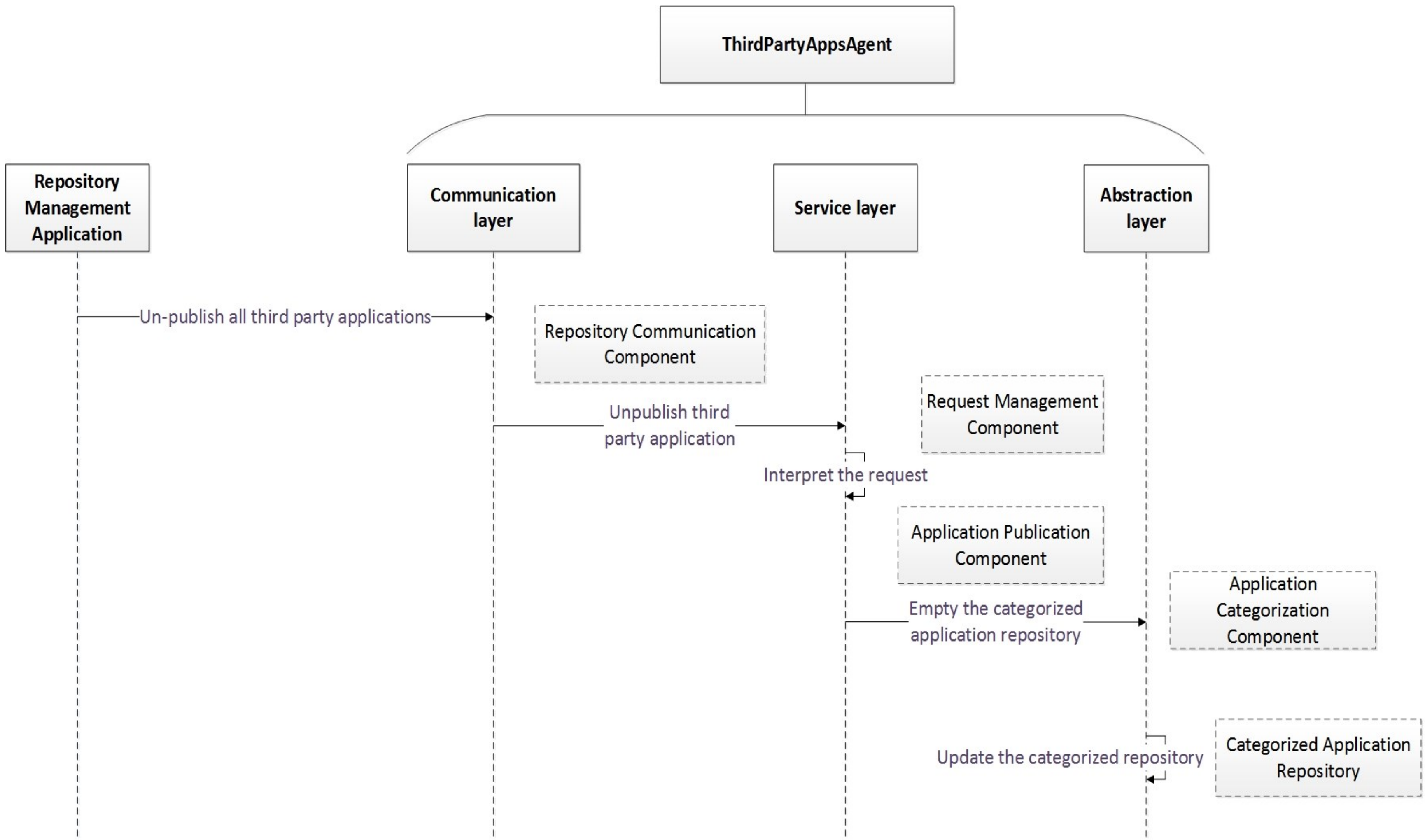

Figure 5-7: The voluntary departure procedure of the ThirdPartyAppsAgent node

5.3.2 Functional Procedures

The functional procedures include the publication, the discovery, the activation and the execution procedures. In this sub-section we describe each procedure using sequence diagrams. For the sake of simplicity, we include only the layers in the sequence diagrams, not the components of each layer.

5.3.2.1 Publication Procedure

The publication procedure describes both the third party application publish and un-publish operations. Figure 5-8 shows the sequence diagram of the publication request. When the *ThirdPartyAppsAgent* node receives a publication request from the management application of the third party application repository, it updates its abstraction layer with the new third party application information.

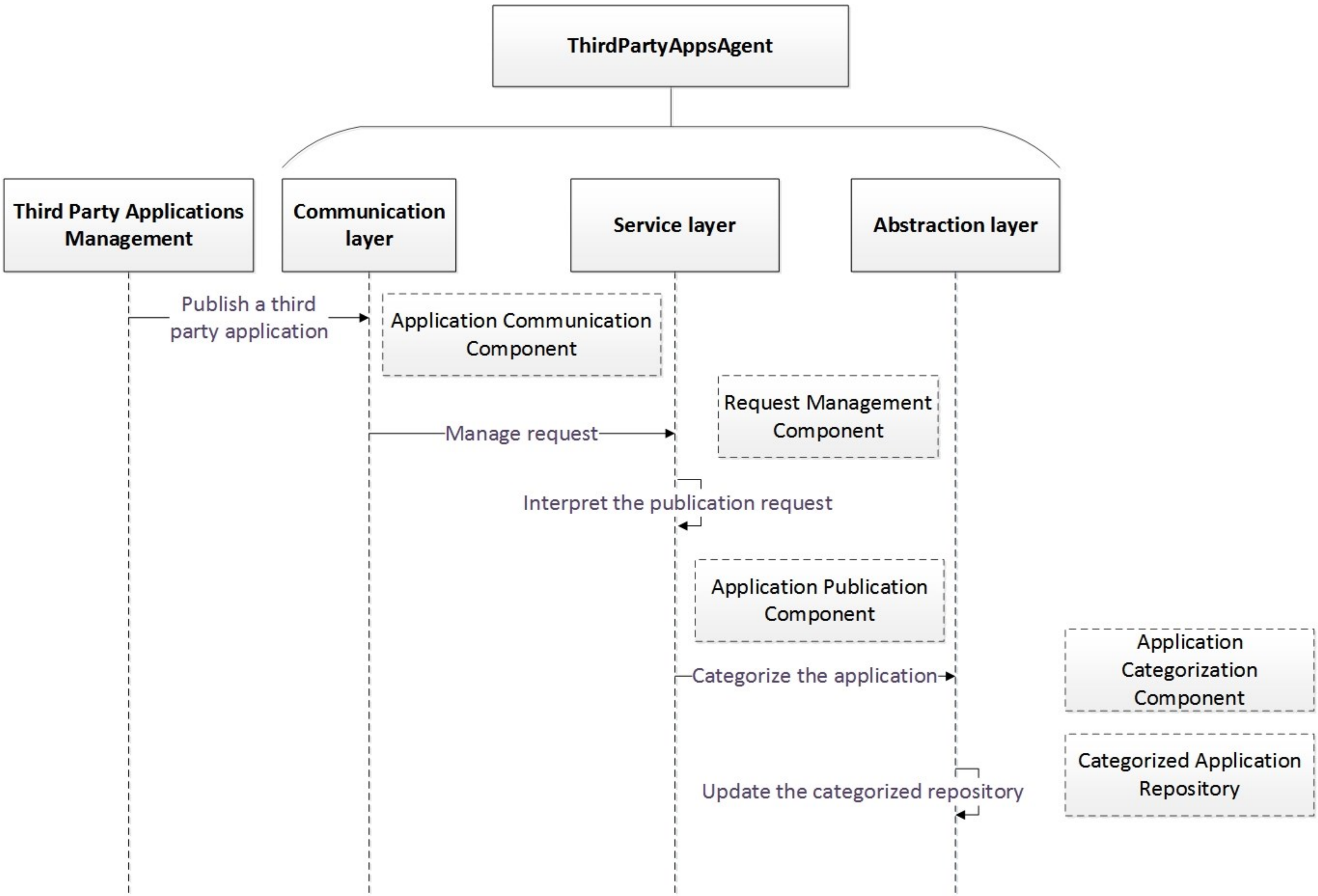

Figure 5-8: The sequence diagram of the publication procedure

5.3.2.2 Discovery Procedure

Figure 5-9 shows a sequence diagram of the discovery request. First, the robotic platform sends a third party application's discovery request to the *RoboticAppsAgent* node that represents it in the overlay. The *RoboticAppsAgent* node maps the request to the ontology using the common application ontology and sends a discovery requeset in the overlay. When a *ThirdPartyAppsAgent* node receives the discovery request, it looks up for the requested third party application in its abstraction layer. Then, it sends back the result of the lookup to the *RoboticAppsAgent* node. If *ThirdPartyAppsAgent* has an application that fits the criteria in the discovery, the information of that application are sent in the discovery response. The *RoboticAppsAgent* sends a list to the robotic platform comprising the discovered third party applications. The robotic platform is responsible for choosing the third party application according to its criteria.

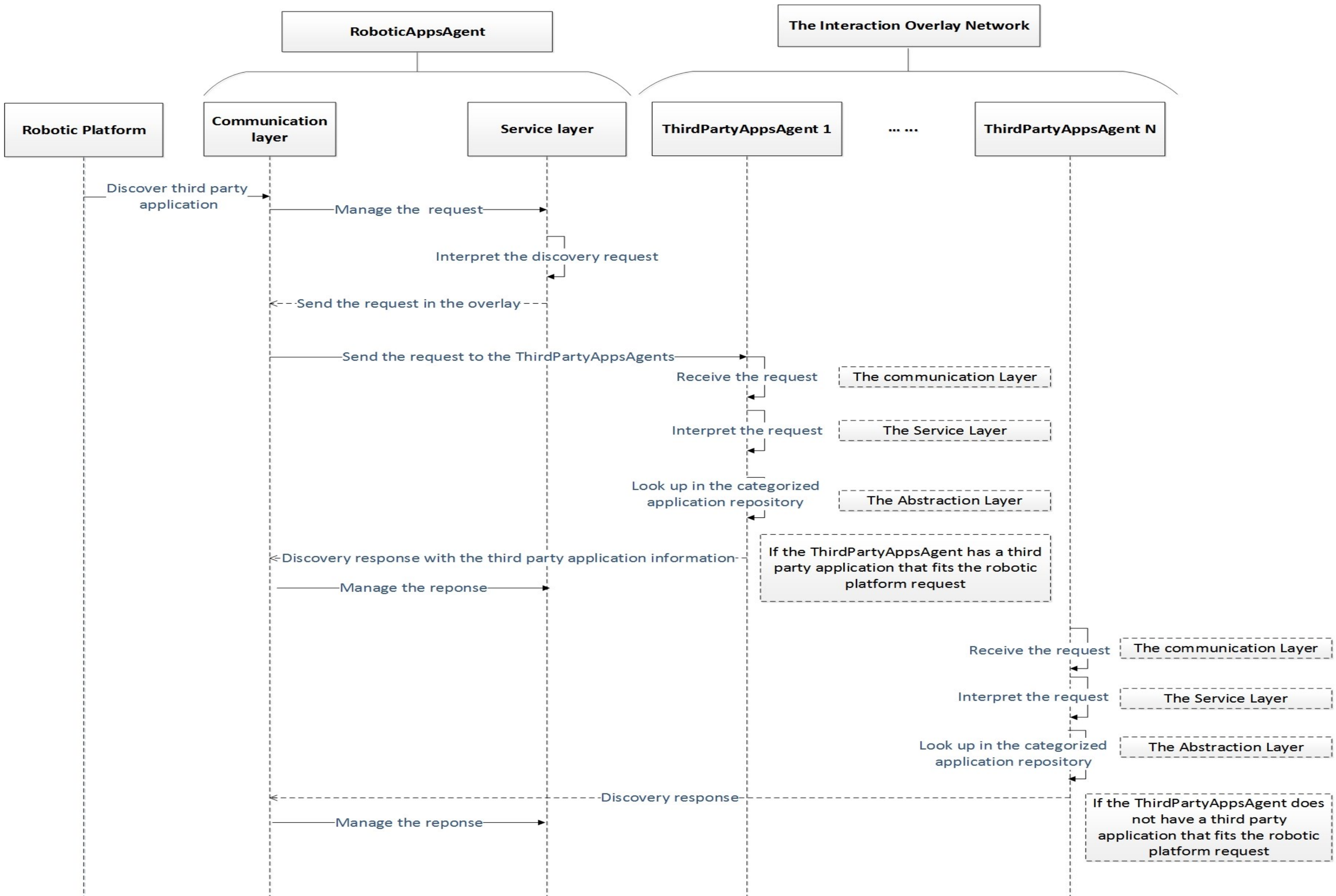

Figure 5-9: The sequence diagram of the discovery procedure

5.3.2.3 Activation Procedure

Figure 5-10 shows a sequence diagram of the activation request. First, the robotic platform sends a third party application activation request to the *RoboticAppsAgent* node that represents it in the overlay. The *RoboticAppsAgent* node interprets the activation request and sends the activation request to the *ThirdPartyAppsAgent* node that represents the third party application in the overlay. When the *ThirdPartyAppsAgent* node receives the request, it sends the activation request to the third party application. The third party application sends the activation response to the *ThirdPartyAppsAgent* node, which in turn sends the response to the *RoboticAppsAgent*.

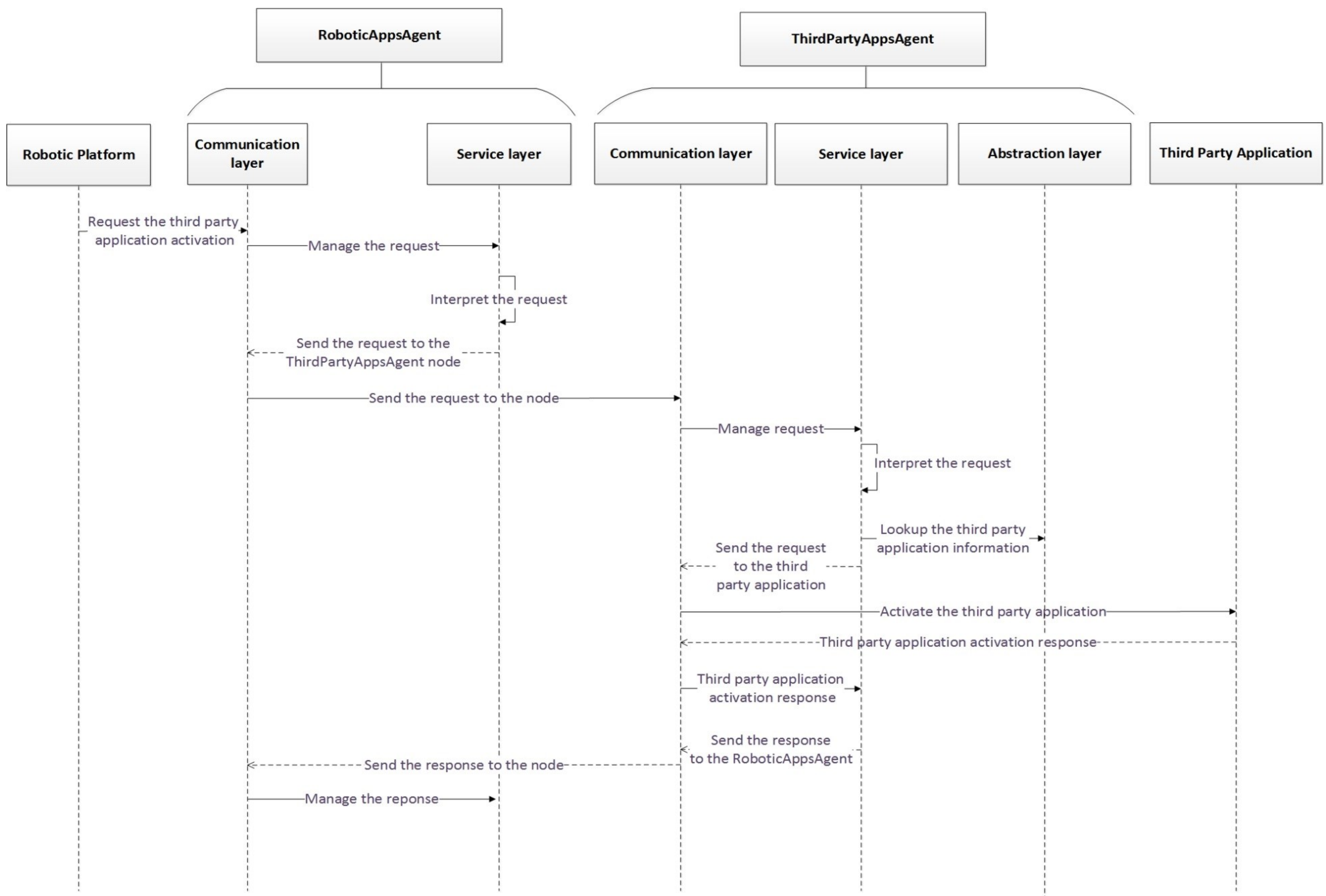

Figure 5-10: The sequence diagram of the activation procedure

5.3.2.4 Execution Procedure

Figures 5-11 and 5-12 show a sequence diagram of the execution request. First, the robotic platform sends and execution request/subscription request to the *RoboticAppsAgent* node that represents it in the overlay. The *RoboticAppsAgent* node interprets the execution request. Then, it sends the execution request/notification request to the *ThirdPartyAppsAgent* node that represents the third party application in the overlay. When the *ThirdPartyAppsAgent* node receives the request, it sends an execution request to the third party application. The third party application sends the execution response/notification to the *ThirdPartyAppsAgent* node. The latter sends the execution response/notification to the *RoboticAppsAgent*.

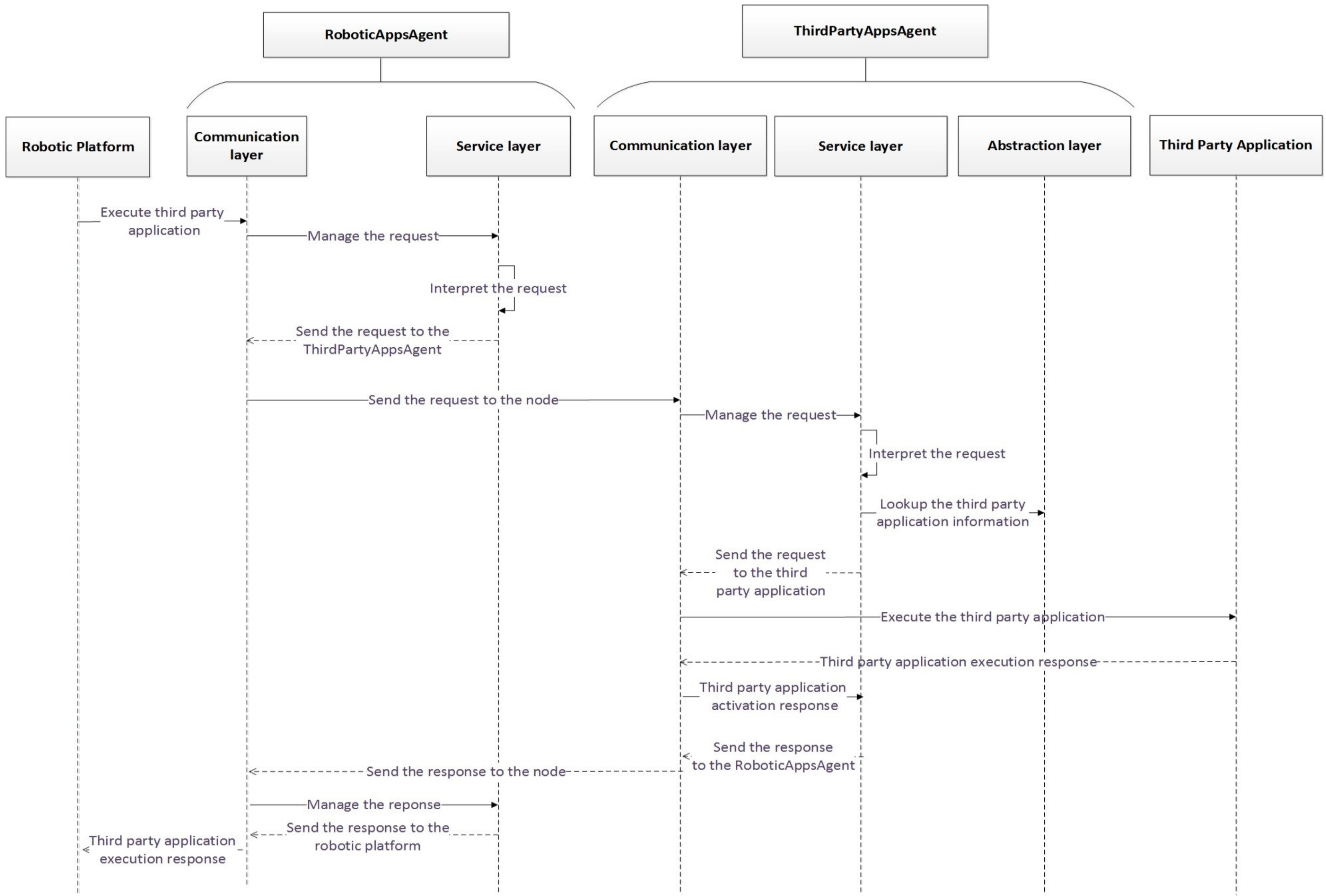

Figure 5-11: The sequence diagram of the execution procedure (1)

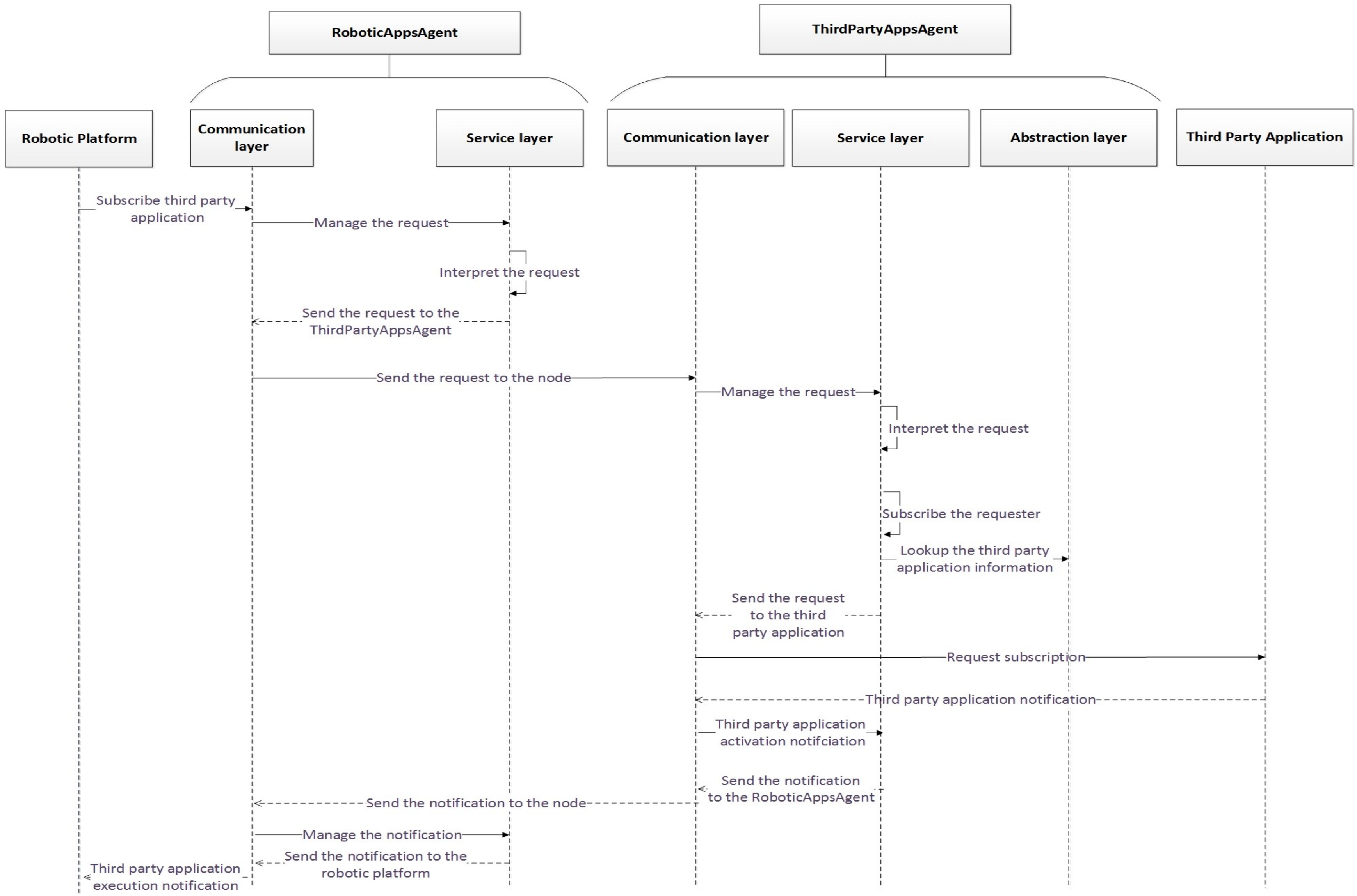

Figure 5-12: The sequence diagram of the execution procedure (2)

5.4 The Wildfire Suppression Use Case

In this section, we apply the proposed architecture to the wildfires suppression application domain. First, we start by identifying the architectural elements. Then, we describe an example of a common ontology that can be used.

5.4.1 The Architectural Elements

In the wildfire suppression application domain, we can identify the following elements: the robotic platform used to provide the wildfire suppression application, the fire detection and the supply management applications with their respective repositories, and the interaction overlay network.

The interaction overlay network contains the following nodes:

- two *ThirdPartyAppsAgent* nodes that represent respectively the fire detection and the supply management applications,
- one *RoboticAppsAgent* node that represents the robotic platform,
- and an *OntologyStore* node that contains the ontology that describes the wildfire suppression, the fire detection, and the supply management domains

5.4.2 The Common Ontology

To the best of our knowledge, there is no existing ontology that already covers the three domains. To build such ontology, we can either identify ontologies for each domain and merge them, or choose an ontology that contains most of the concepts and extend it. In this thesis, we chose the second alternative. We use the OntoFire ontology [35]

] as a starting point and add the missing concepts. In the next sub-sections, we first describe the OntoFire, then, we present the new concepts and relationships we added to it. This will constitute the common ontology for the wildfire suppression. Finally, we describe examples of requests that can be done using the common ontology.

5.4.2.1 OntoFire

The OntoFire ontology is shown in Figure 5-12. OntoFire ontology is designed to describe the semantic context of wildfires and their associated risks for geo-portal navigation use. The core concepts of the OntoFire are "natural risk", "hazard", and "vulnerability". Wildfires risk is assessed based on hazard and vulnerability of both physical and socioeconomic environments. "Hazard" is the result of interaction between the "physical environment" and "natural risks". "Vulnerability" is expressed as the relationship between "natural risk" and "infrastructure".

"Natural risks" and "physical environment" concepts are related to both fire detection and wildfire suppression application domains. "Natural risks" can be categorized in into "climatic" and "geological" (e.g, "earthquakes"). "Climatic" risks are specialized to "atmospheric" (e.g. "storms"), "hydrological" (e.g. "floods" and "draughts"), and "biophysical" (e.g. "fires") natural risks. "Meteorology and Climate", "Topography" and "Vegetation" are concepts affiliated to "Physical environment".

"Infrastructures" concept is related to wildlife suppression application and supply management application domains. "Infrastructures" may be "Fire Management" objects (i.e. "firefighting outposts" and "firefighting vehicles").

It has a generic class "Geo-Information Resource" that holds properties of the ISO metadata standard for geo-portals. This standard contains the required schema to describe geographic information and services. It contains information about the identification and spatial reference of digital geographic data. Every resource classified under any class of the OntoFire ontology is a "Geo-Information Resource".

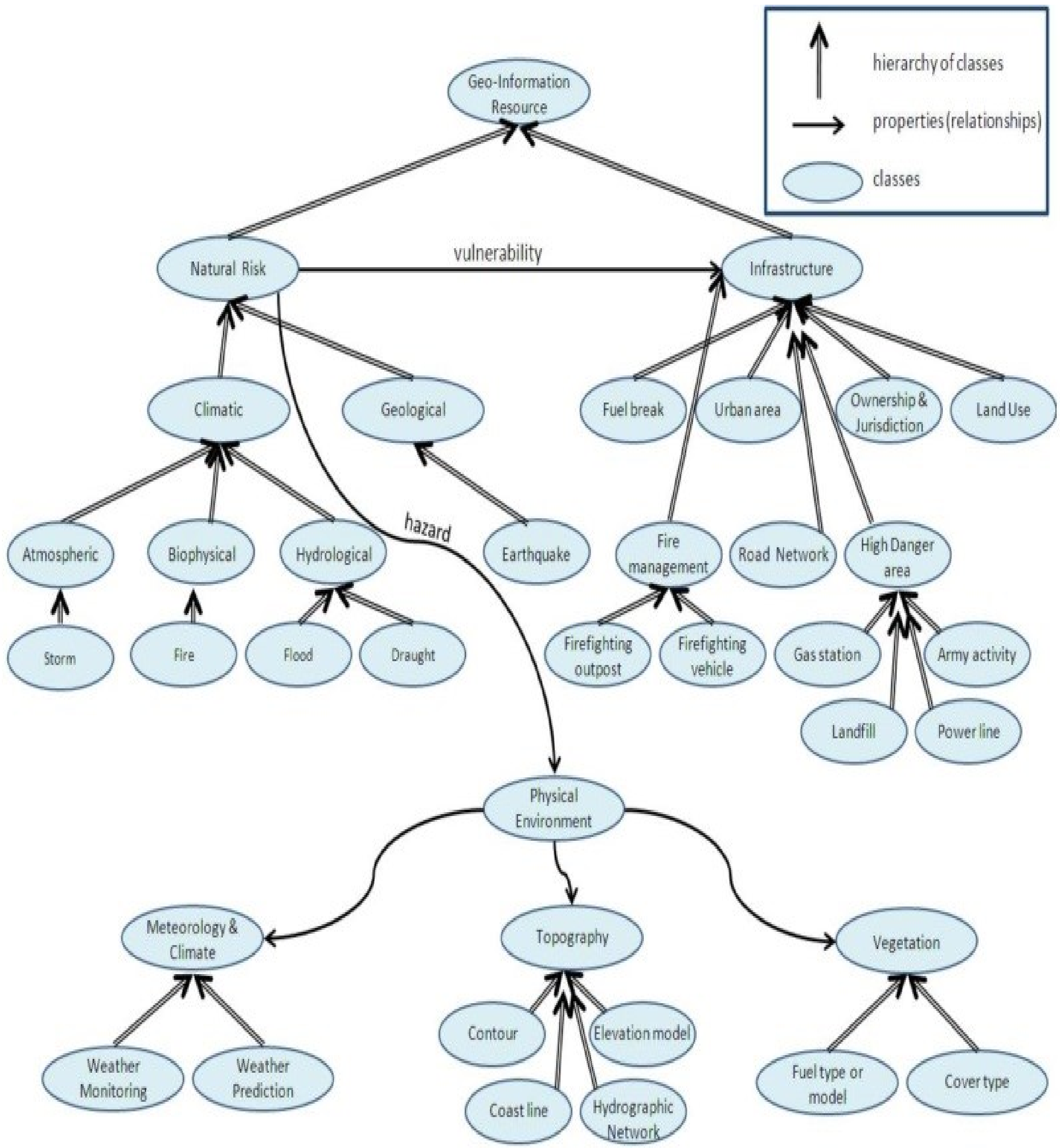

Figure 5-12: The OntoFire ontology [35]

5.4.2.2 OntoFire Extension

Figure 5-13 shows the added concepts and relationships related to the fire detection, supply management, and wildfire suppression domains. Since every concept in the OntoFire ontology is a specialization of "Geo-Information Resource", we needed to add a more generic class "Thing" from which we can derive concepts that are not "Geo-Information Resource". The class "Thing" is empty, and it is a generalization of "Geo-Information Resource". For the three domains, we added the “application” concept as a specialization of “thing”, and from which we derive “fire detection” and “supply management”. The “application” concept has two properties: “name” and “activation interface”, and “execution interface”. The activation and execution interfaces are a specialization of “thing “. The “activation interface” has “name” and “arguments” as properties. The “execution interface” has “name”, “arguments”, and, “return values” as properties. “Arguments” and “return values” are specialization of “thing”.

For the wildfire suppression and the supply management application domains, we added “robot” concept (e.g., “aerial robots”, “ground robots”) as “fire management” objects. We also added “fire retardant” concept (e.g. “foam” and “water”) which is a “thing” and not a “Geo-Information Resource”. We added the “supply station” concept as a specialization of “firefighting outpost” object. We also added the relationship between the supply stations and the retards, because “supply stations” have a supply of type “retardant” (“water” or/and “foam”). “Robots” use “supply stations”. The “wildfire suppression” application manages “robots” and suppresses “fires”. The “supply management” application manages “supply stations”.

For the fire detection application domain, we added “wireless sensor network” concept that is a “Geo-Information Resource”. A “wireless sensor network” contains “sensors” with different “capabilities” (e.g., “temperature”, “gas concentration”, “moisture”). A “wireless sensor network” is deployed in a “physical environment”. The “fire detection” application uses a “wireless sensor network” to trigger “fire notifications” that correspond to “fires”.

Figure 5-13: The OntoFire extension

5.4.2.3 Request Examples

In this sub-section, we describe some of the requests that can be done using the Ontofire extended ontology.

Publication requests

If a supply management application called “Water Supply Management” that manages supply stations with water retardant wants to publish itself, it will send a publication request to its *ThirdPartyAppsAgent* node that contains the following information:

Add “Supply Management”

Where:

“Supply management” has property “name” with value “Water Supply Management”

“Supply management” has property “activation interface” with value ‘name:“activate”, argument ””’

“Supply management” has property “execution interface” with value ‘name:“GetClosestSupplyStation”, argument ”Robot”, return value “Supply Station” ’

“Supply management” has property “name” with value “Water Supply Management”

And

“Supply Management” manages “Supply Stations”

Where “Supply stations” has retardant type “water”

Discovery requests

If the wildfire suppression application wants to discover a fire detection application deployed in an area with a certain vegetation type and the fire detection uses the gas concentration to detect fires, it will send a discovery request to its *RoboticAppsAgent* node containing the following information:

Select "Fire detection"

Where "Fire Detection" uses "wireless sensor network"

And

"Wireless sensor network" is deployed in

Select "physical environment"

Where "vegetation type" has value "Boreal Mixedwood"

And "wireless sensor network" contains

Select "sensor"

Where "sensor" has "capability" of type "gas concentration"

5.5 Chapter Summary

In this chapter, we described the proposed architecture which is based on P2P overlay networks. We presented the overall architecture, including the architectural principles and the architecture overview. Then, we describe the architecture in details. We presented the overlay protocols. Next, we described nodes architecture, which comprises three layers: communication, service, and abstraction layers. Inside each layer there are components that are common to every node and node-specific components. We describe each of these components. Afterwards, we described the overlay procedures. We started with the management procedures: joining and departure. Then, we described the functional procedures: publication, discovery, activation, and execution. Finally, we used the wildfire suppression application to showcase the use of ontologies in the proposed architecture.

6 Validation: Prototype and Evaluation

To validate the proposed architecture, we designed and implemented a prototype based on the wildfire suppression application domain. The first section presents the overall prototype architecture. The second section describes in details the prototype architecture, which includes the implemented applications, and the interaction overlay network. The third section includes the prototype setup, the end-to-end execution, and the performance measurements.

6.1 Overall Prototype Architecture

In this sub-section, we will start by presenting the implemented scenario. Then, we give a high level description of the prototype. Finally, we describe the software tools we used in the implementation.

6.1.1 Implemented Scenario

We have implemented the wildfire suppression scenario. We consider a forest where wireless sensor networks have been deployed to monitor wildfires. A fire detection application manages the wireless sensor networks. When there is a fire breakout, the fire detection application sends a notification to the wildfire suppression application. The wildfire suppression application has a fleet of firefighting robots composed of aerial and ground robots. These firefighting robots collect fire retardants from supply stations. These supply stations are managed by a supply management application. When the wildfire suppression application receives a fire notification, it requests the supply management application for the most adequate supply stations where it can deploy its robots. It sends the robots to the supply stations to collect fire retardants, then to the fire location to suppress the fire. Each application (fire detection, supply management, and wildfire suppression) is offered by a distinct provider.

6.1.2 Prototype High Level Description

Figure 6-1 shows the fire detection interface offered to end-users. To simulate a fire notification, we used a map of Montreal with six highlighted locations that represent where fire breakouts are monitored. When the user clicks on one of these locations, it triggers a fire notification with the fire location.

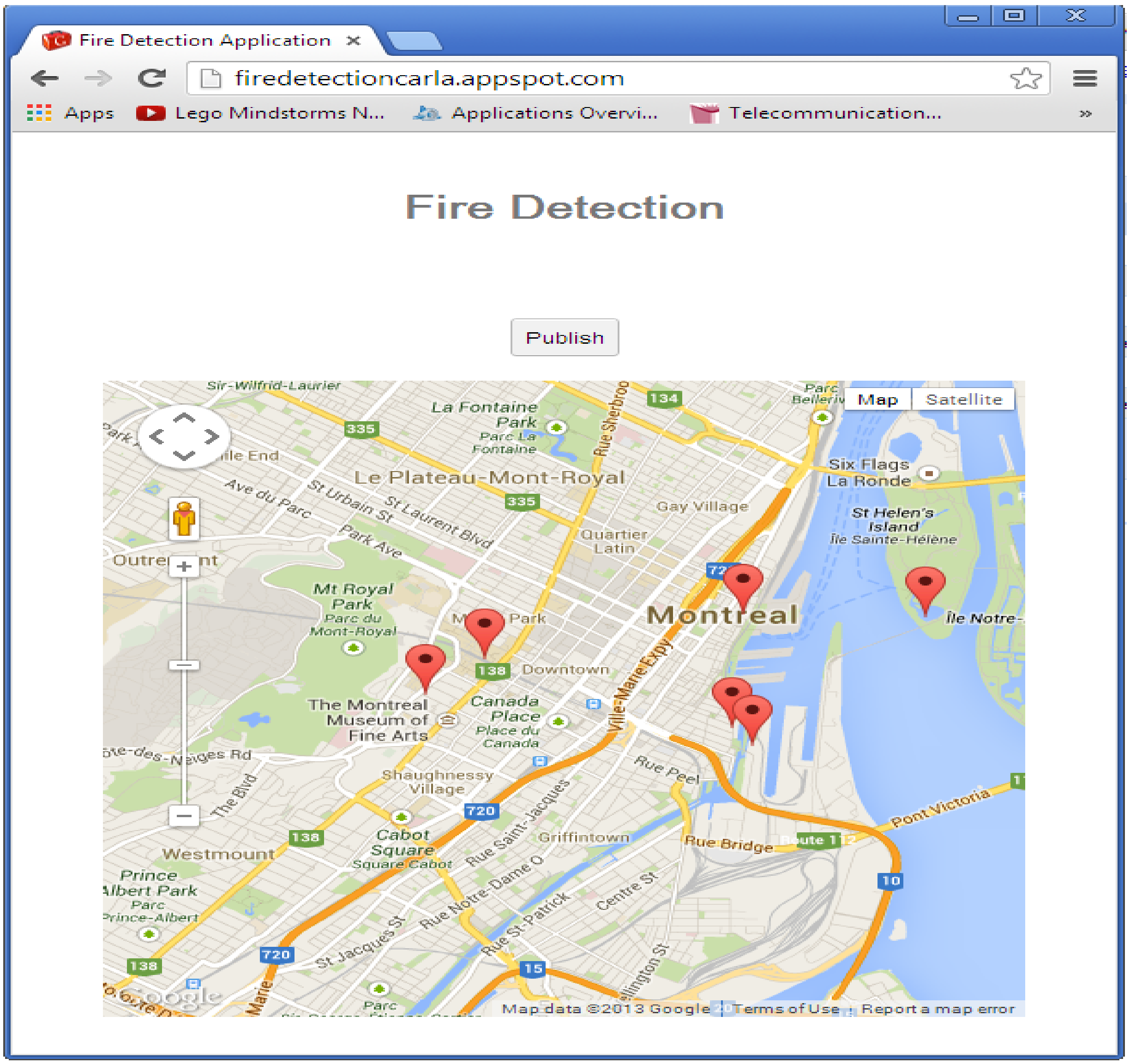

Figure 6-1: The fire detection application end user interface

The supply management application manages three supply stations that are randomly located in Montreal area.The wildfire suppression application controls one LEGO robot. It maps fire locations and supply locations that it receives to pre-defined landmarks that we used in the laboratory. The retardant supplies that the robot gets from supply stations are simulated using plastic balls that the robot grabs to collect retardants. The fire suppression is simulated by dropping the ball in the fire location.

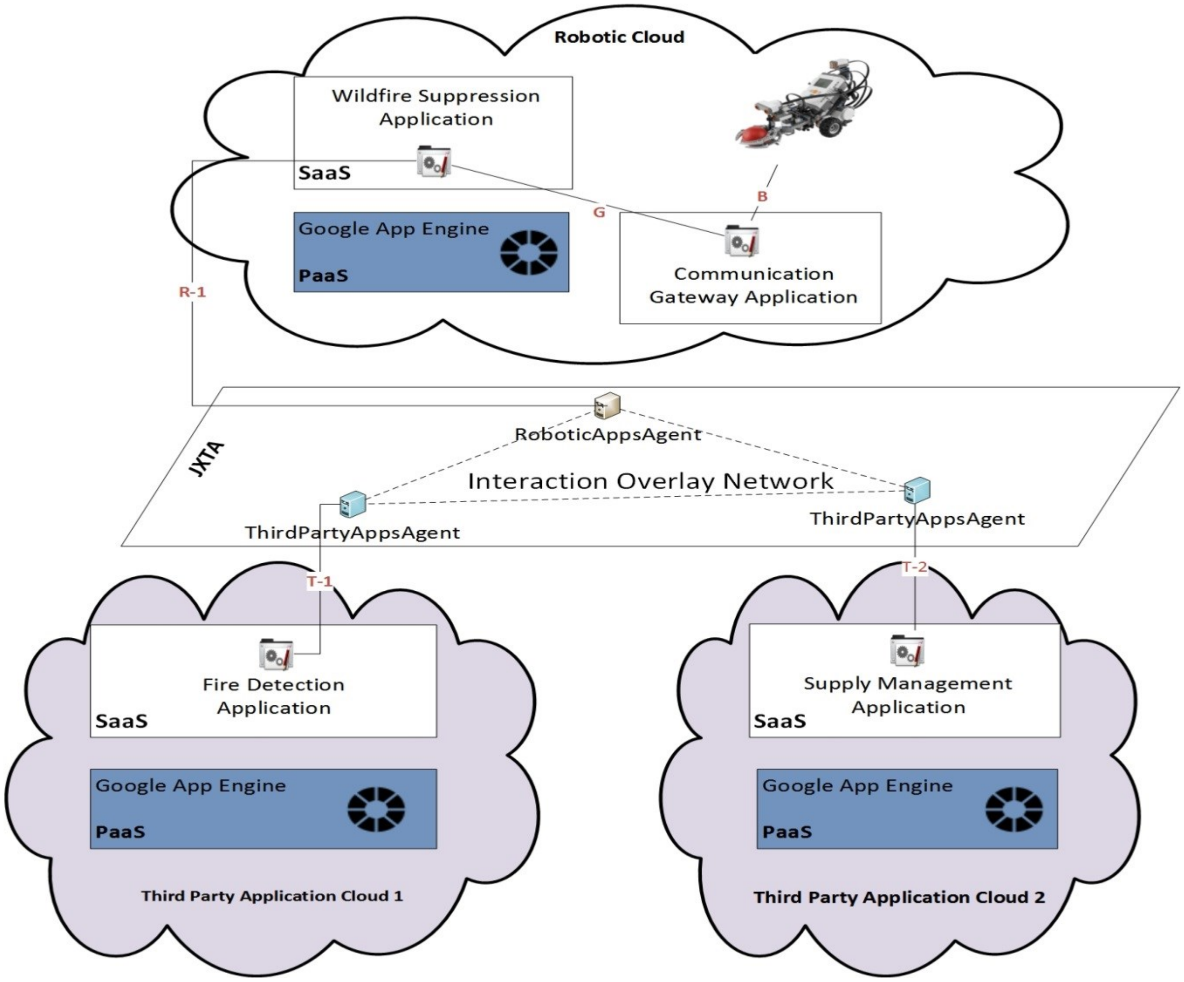

Figure 6-2: The prototype architecture

Figure 6-2 illustrates the prototype architecture. We implemented the two *ThirdPartyAppsAgent* nodes corresponding to the fire detection and the supply management applications. We also implemented the *RoboticAppsAgent* node corresponding to the wildfire suppression application. We did not implement the *OntologyStore* node. T-1 and T-2 refer to the fire detection application interface, and the supply management interface respectively, while R-1- refers to the wildfire suppression application interface. T-1 and R-1 are REST-based, and T-2 is SOAP-based.

We implemented a RESTful application that plays the role of communication gateway between the wildfire suppression application and the robot. G and B refer to REST and Bluetooth interfaces respectively.

6.1.3 Software Tools

In this sub-section, we will describe the tools we used to implement the scenario.

6.1.3.1 Google Apps Engine

We used GAE to implement the wildfire suppression, fire detection and supply management applications. GAE is a platform that allows the development of applications that are hosted on Google Infrastructure [37]. A user may upload applications in GAE and it is ready to use by its customers. The user does not have to maintain any server and has not administration work to do to offer the applications. The applications may be offered at the free domain appspot.com or from a domain that the user may specify. The user decides if the access to the applications is public or restricted to a limited number of members. Users and customers must sign up for a free Google account. The initial package is free of use, it includes up to 500Mb of storage and bandwidth to serve 5 million page views per month [37]. We chose GAE platform to develop the applications because it's easy to deploy and maintain applications.

6.1.3.2 LEGO Mindstorms NXT Robots

We used LEGO Mindstorms NXT robot as the wildfire suppression robot fleet. LEGO Mindstorms NXT robots are modular robots that are programmable and reconfigurable [38]. They are widely used in educational institutes. They offer a graphic-based programming interface for novice developers. They come in modular blocks that have to be assembled. These blocks include motors, sensors, and computing units called LEGO brick. LEGO Mindstorms NXT robots can only communicate via Bluetooth or USB.

We use LEGO Mindstorms NXT robot of type Tribot. Figure 6-3 shows the assembled robot. It is equipped with a sound, a touch, and ultrasound, and a light sensor. It also has two motors to move the robot around and to move the arms to grab objects. The Tribot can grab only plastic balls that come with the robot kit. We chose this robot because of its re-usability; the same robot kit can be assembled into different robots.

Figure 6-3: LEGO Mindstorms NXT robot of type Tribot

6.1.3.3 RESTlet Framework

We used RESTLet framework to develop the RESTful Web services. RESTful web services are designed following REST principles [39]. REST is a technology-independent architectural style for distributed systems. REST principles include: addressability, uniform interface, and statelessness. In REST, datasets are represented to as resources; addressability is the ability to refer to these resources using a unique identifier, called Uniform Resource Identifier (URI). We chose RESTful Web services because they are easy to develop and they offer good scalability [39]. RESTful Web services commonly use HTTP as a transfer protocol.

RESTlet is an open source framework for the Java platform. It provides a mapping between REST concepts and Java classes. An application that is developed with RESTlet can play the role of HTTP server or HTTP client interchangeably without any need of change in the code of the application. We chose RESTlet framework because it is an object-oriented framework, and is fairly simple to use [40].

6.1.3.4 SoapUI

SoapUI is a free functional testing tool for SOAP. It allows to create and execute SOAP client using the WSDL file of the Web service and to test it [41]. SOAP-based Web services are based on SOA principle. They have three entities: service provider, service registry, and service requester [22]. The Web service description contains a detailed description of the methods that the service offers, with their parameters.

6.1.3.5 JXTA

As previously presented in chapter 2, JXTA is an open source project to create structured P2P overlay networks. It defines peers with different communication level capabilities. Super peers can enable edge peers to communicate with firewalls. Peers may form a group depending on the services they share. Peers can communicate through pipes which can be unidirectional or bidirectional. Peers, groups, services, and pipes are described using XML files called advertisements.

6.2 Detailed Prototype Architecture

In this sub-section, we start by presenting the application software architecture. Then, we describe the overlay nodes software architecture. Next, we describe the fire detection, the supply management, and the wildfire suppression interfaces. Then, we describe the messages exchanged between the nodes in the overlay. Finally, we describe the overlay procedures.

6.2.1 Third Party Applications and Robotic Application Software Architecture

The three applications contain a web service requester and a web service provider. The web service provider is used to offer services to end-users and receive requests/responses from the overlay node corresponding to the application. The web service requester is used to send requests to the overlay node corresponding to the application. For the wildfire suppression and the fire detection applications, the web service provider is developed as a RESTful web service. The web service requester is a REST client. For the supply management application, the web service provider is SOAP-based and the web service request is an HTTP client.

6.2.2 Overlay Nodes

Figure 6-4 shows the software modules and interactions of the *RoboticAppsAgent* node. The web service provider was developed as a RESTful web service using RESTlet framework. The web service provider is a REST client. The request management, application discovery, and application execution components were developed as JAVA classes and methods.

Figure 6-5 shows the software modules and interactions of the *ThirdPartyAppsAgent* node corresponding to the fire detection application. We used the same web service provider for the application communication component and the repository component. The web service provider was developed as a RESTful web service using RESTlet framework. The web service requester is a REST client. The request management, application discovery, application publication, and application execution components were developed as JAVA classes and methods

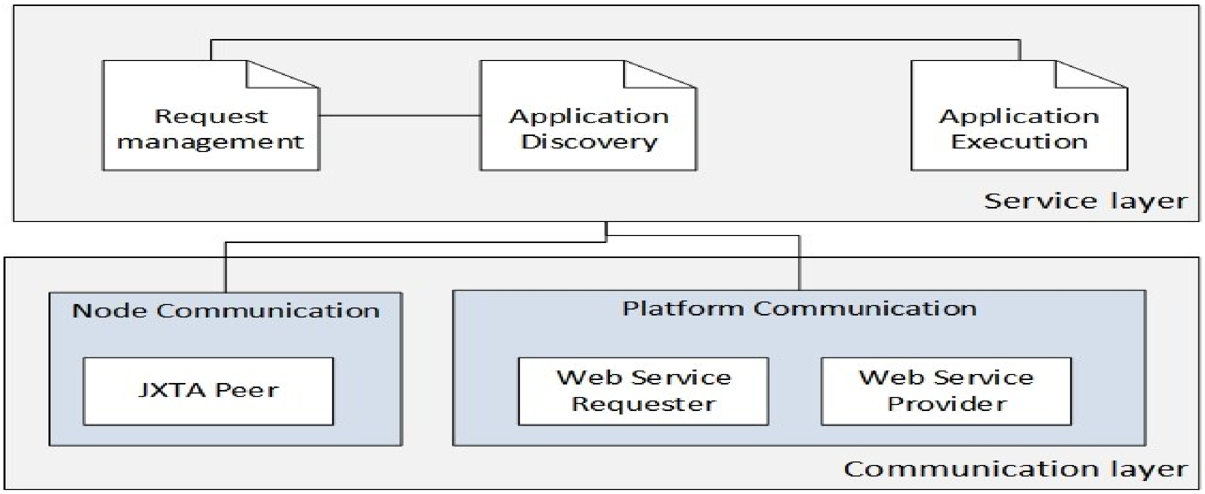

Figure 6-4: The software modules and interactions of the RoboticAppsAgent node

.

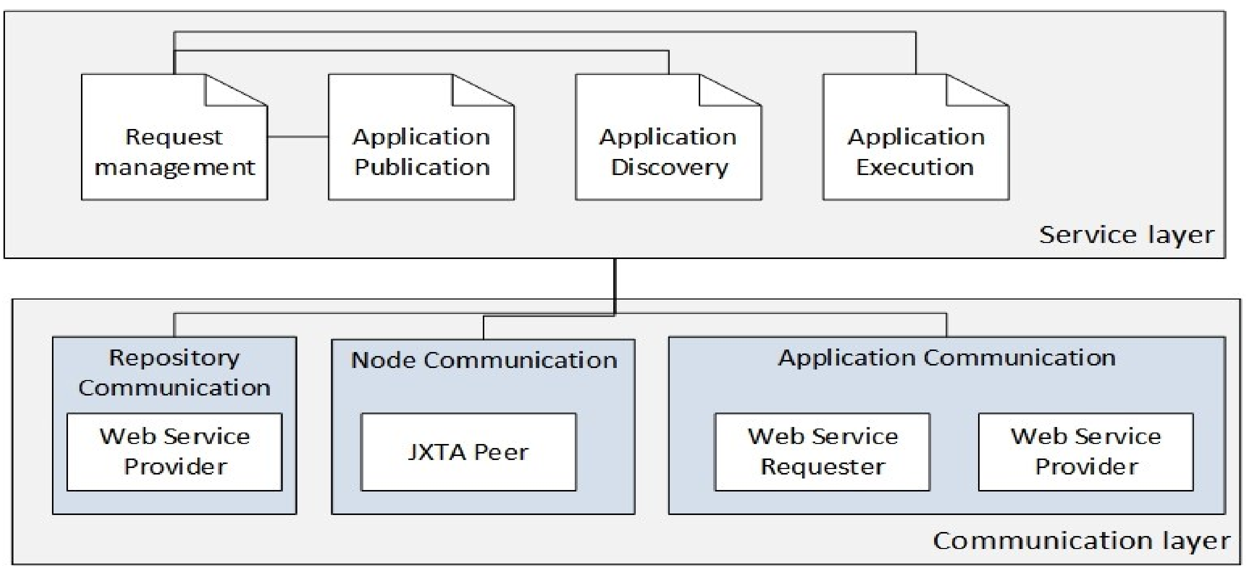

Figure 6-5: The software modules and interactions of the fire detection ThirdPartyAppsAgent node

Figure 6-6 shows the software modules and interactions of the *ThirdPartyAppsAgent* node corresponding to the supply management application. The web service provider was developed as a RESTful web service using RESTlet framework. The web service requester was generated using SoapUI. The request management, application discovery, application publication, and application execution components were developed as JAVA classes and methods.

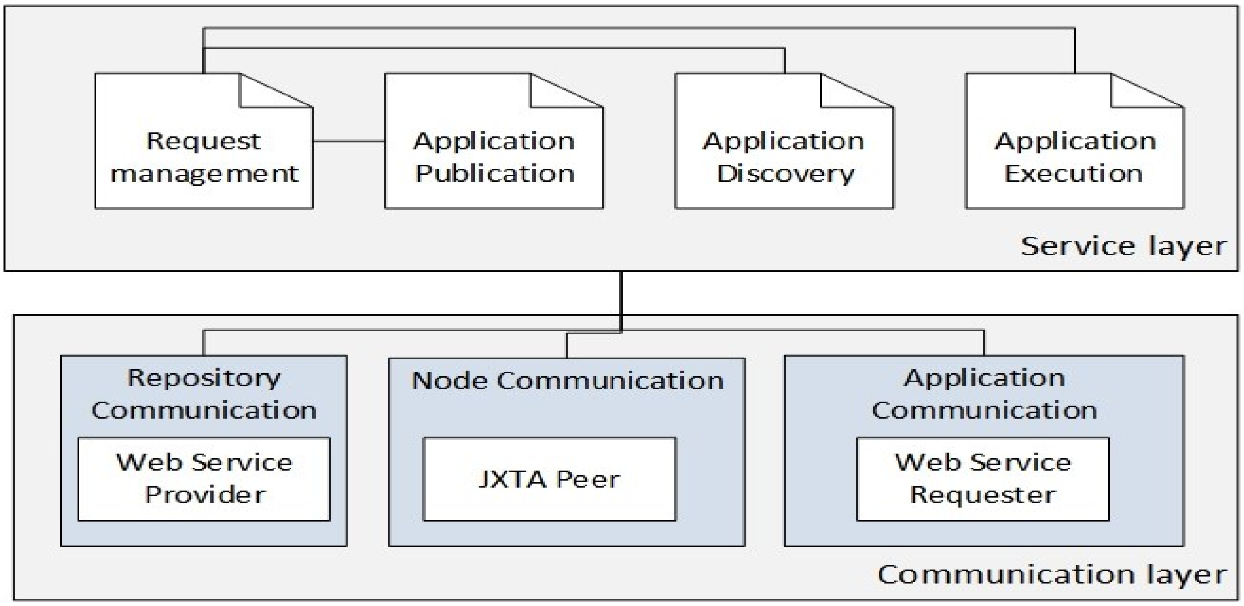

Figure 6-6: The software modules and interactions of the supply management ThirdPartyAppsAgent node

The publication of the two applications is carried out by a JXTA advertisement. The advertisement holds the name of the third party application (i.e., fire detection and supply management). JXTA advertisements allow each node to expose information for the other nodes in the overlay network to discover.

The execution messages of the third party applications in the overlay were mapped to JXTA messages that are exchanged through JXTA bidirectional pipes.

6.2.3 Interfaces

We start by describing the interface between the wildfire suppression application and its corresponding *RoboticAppsAgent* node. Then, we describe the interface between the fire detection application and its corresponding *ThirdPartyAppsAgent* node, and between the supply management application and its corresponding *ThirdPartyAppsAgent* node.

6.2.3.1 The Interface Between the Wildfire Suppression Application and the RoboticAppsAgent Node

The RESTlet web service of the wildfire suppression application has two resources: fire notification and getSupplyStationResponse. The *RoboticAppsAgent* node invokes the fire notification resources when it sends the fire notification coming from the fire detection. *RoboticAppsAgent* node invokes the getClosestSupplyStationResponse resources when it receives the response to the getClosestSupplyStation request. Table 3 describes this resource.

Table 3: The wildfire suppression application resource description

Resource	Operation	HTTP action
Notification	Create a fire notification with the location of where the fire occurred	POST
getClosestSupplyStation Response	Create a response to the getClosestSupplyStation request	POST

The RESTlet web service of the *RoboticAppsAgent* node has two resources: discovery and closest supply location. The wildfire suppression application invokes these resources when it discovers applications and when it requests the closest supply station to a given fire location. Table 5 describes these resources.

Table 4: The RoboticAppsAgent node resource description

Resource	Operation	HTTP action
Discovery	Create a discovery request for the third party applications in the overlay network	POST
getClosestSupplyStation	Creates an execution request with the location where the fire occurred	POST

6.2.3.2 The Interface Between the Fire Detection Application and the ThirdPartyAppsAgent Node

The RESTlet web service of the *ThirdPartyAppsAgent* node has two resources: publication and fire notification. The fire detection application invokes these resources when it publishes itself in the overlay network and when the fire detection triggers a fire notification. Table 6 describes these resources.

Table 5: The fire detection ThirdPartyAppsAgent node resource description

Resource	Operation	HTTP action
Publication	Create a publication request of the fire detection application	POST
FireNotification	Create a fire notification that will be sent to the wildfire suppression application	POST

6.2.3.3 The Interface Between the Supply Management Application and the ThirdPartyAppsAgent Node

The SOAP server of the supply management application exposes one method: getClosestSupplyLocation. This method takes two arguments: the latitude and the longitude of a requested location. GetClosestSupplyLocation will return the closest supply location to the reference location.

The SOAP client of the *ThirdPartyAppsAgent* node invokes the getClosestSupplyLocation method when it receives a request from the *RoboticAppsAgent* node to get the closest supply location to a given fire location.

6.2.4 Procedures

We implemented the publication, discovery and execution procedures. We will describe an example of each of these procedures in this sub-section.

6.2.4.1 Publication Procedure

Figure 6-7 shows the sequence diagram of the publication procedure for the fire detection application.

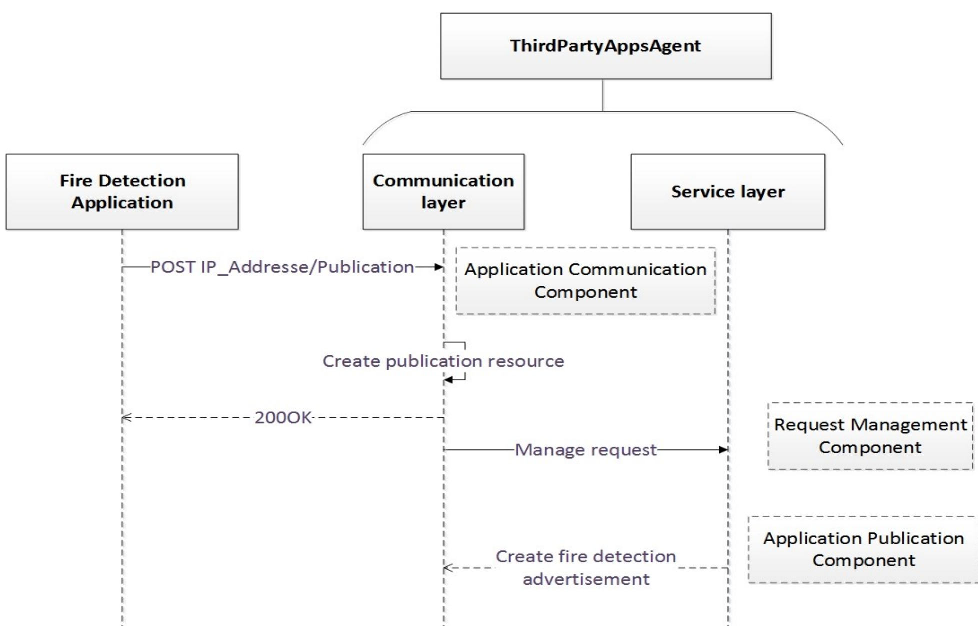

Figure 6-7: The sequence diagram of the publication procedure of the wildfires suppression scenario

6.2.4.2 Discovery Procedure

The discovery procedure is triggered from the wildfire suppression application. Its sequence diagram is illustrated in Figure 6-8. The discovery is carried out by a discovery listener of JXTA's advertisements. The discovery listener of the *RoboticAppsAgent* node looks for advertisements that hold third party applications names (i.e., fire detection or supply management).

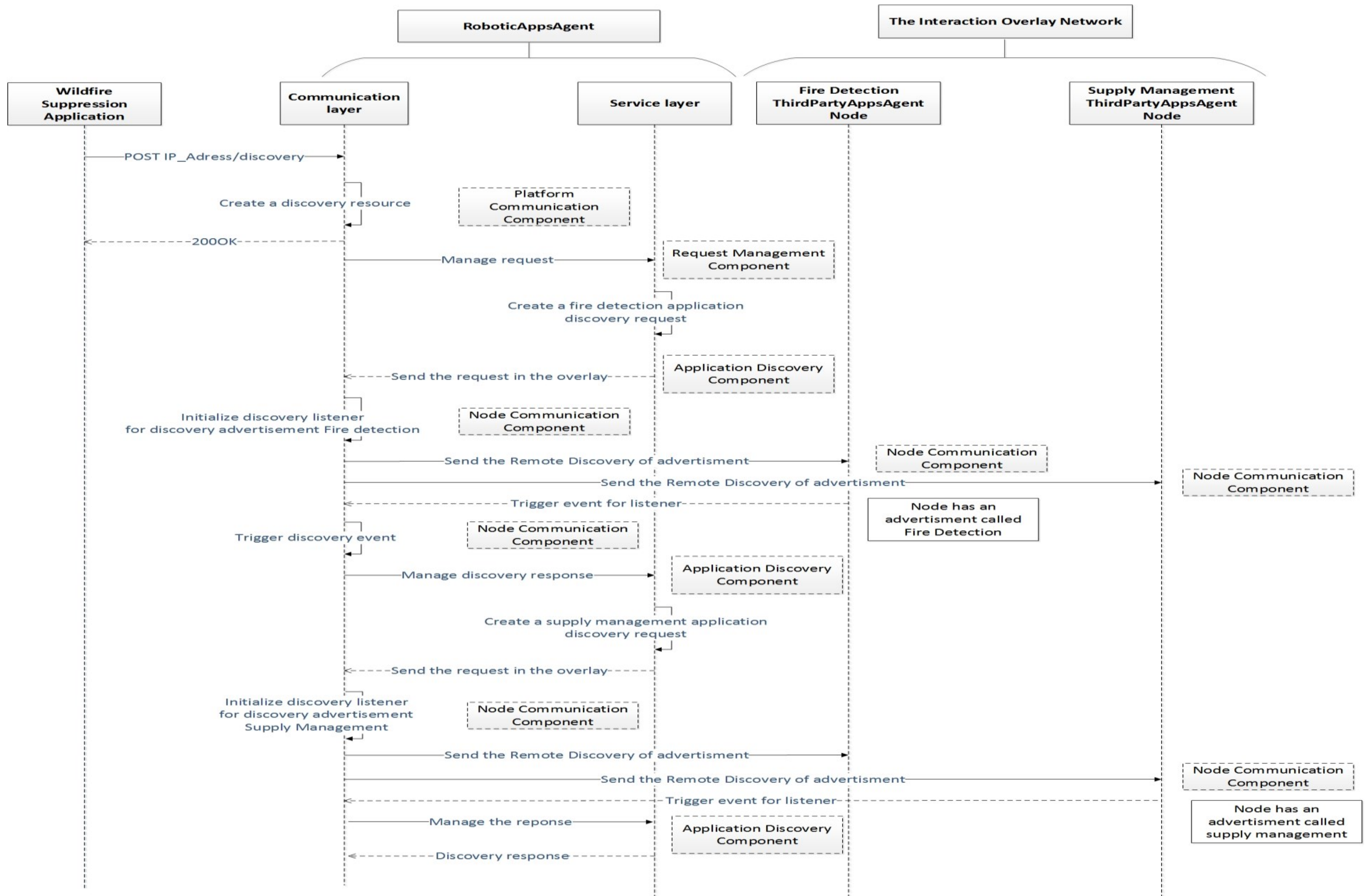

Figure 6-8: The sequence diagram of the discovery procedure of the wildfires suppression scenario

6.2.4.3 *Execution Procedure*

The execution procedures include the fire notification and the closest supply location. For both the fire detection and the supply management application, the *RoboticAppsAgent* node opens a JXTA bidirectional pipe with the *ThirdPartyAppsAgent* node after it discovers the third party application. The execution messages are carried out JXTA bidirectional pipes. Figure 6-9 shows a sequence diagram of the closest supply location execution procedure. The fire notification procedure will be included in the end-to-end execution.

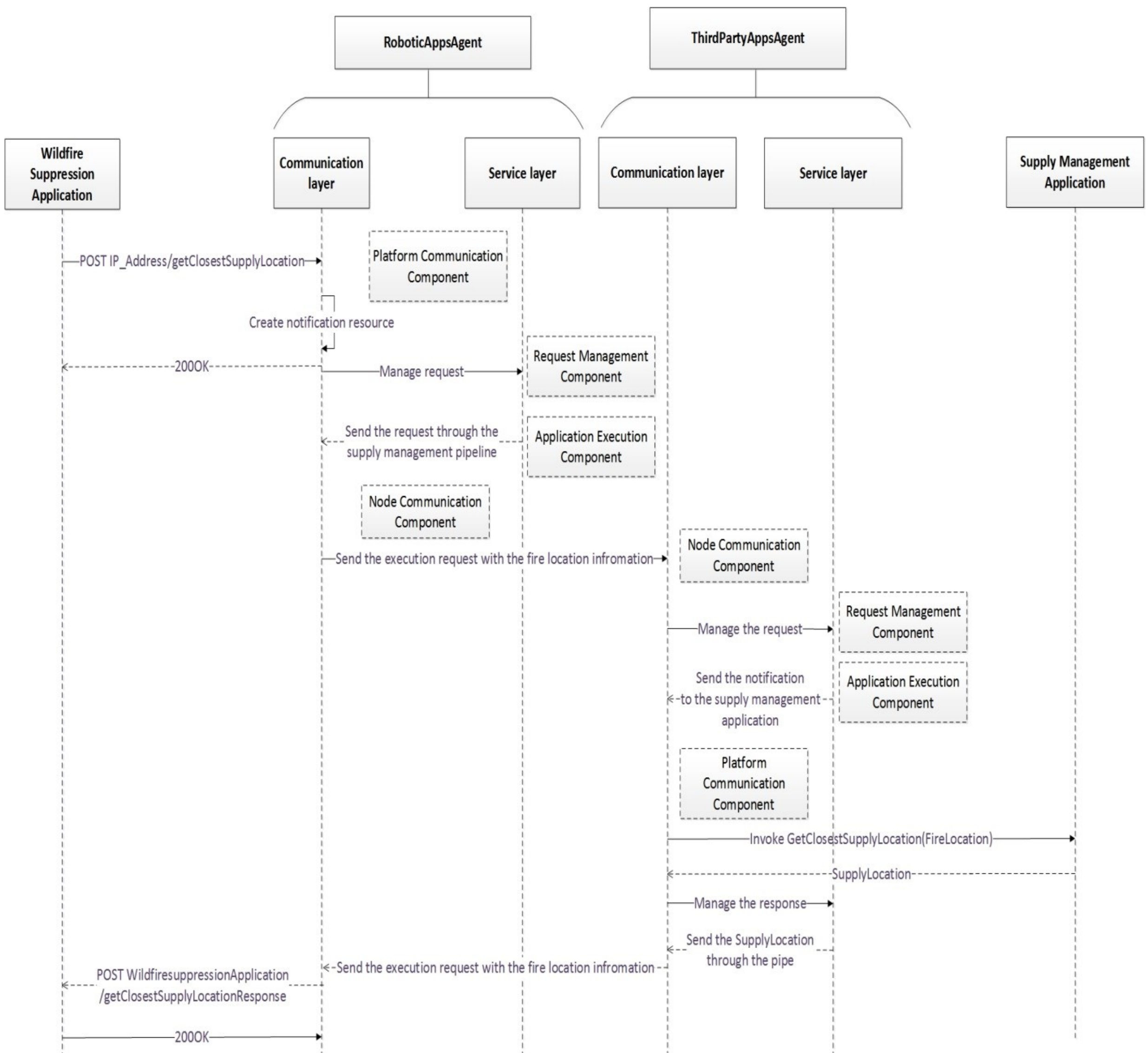

Figure 6-9: The sequence diagram of the execution procedure of the wildfire suppression scenario

6.3 Prototype Setup, End-to-End execution and Performance Measurement

To execute the prototype, we used three laptops. In this sub-section, we describe the networking choices we made and the details of the setup. Then, we present the end-to-end execution of the prototype. Finally we present the performance measurements.

6.3.1 Prototype Setup

Figure 6-10 shows the prototype setup. The fire detection, supply management and the wildfire suppression applications are hosted on Google Infrastructure. The *RoboticAppsAgent* node is executed on a laptop, while the two *ThirdPartyAppsAgents* nodes are executed on a two other laptops.

We connected the three laptops in the same Local Area Network (LAN). One of the laptops has two network interfaces, one where it has a public IP, the other one is the LAN interface. The requests coming from the wildfire suppression, the fire detection, and the supply management applications are send to the public IP. We developed a Network Address Translation (NAT) server as a RESTLet application. The NAT application redirects the requests coming to the overlay nodes.

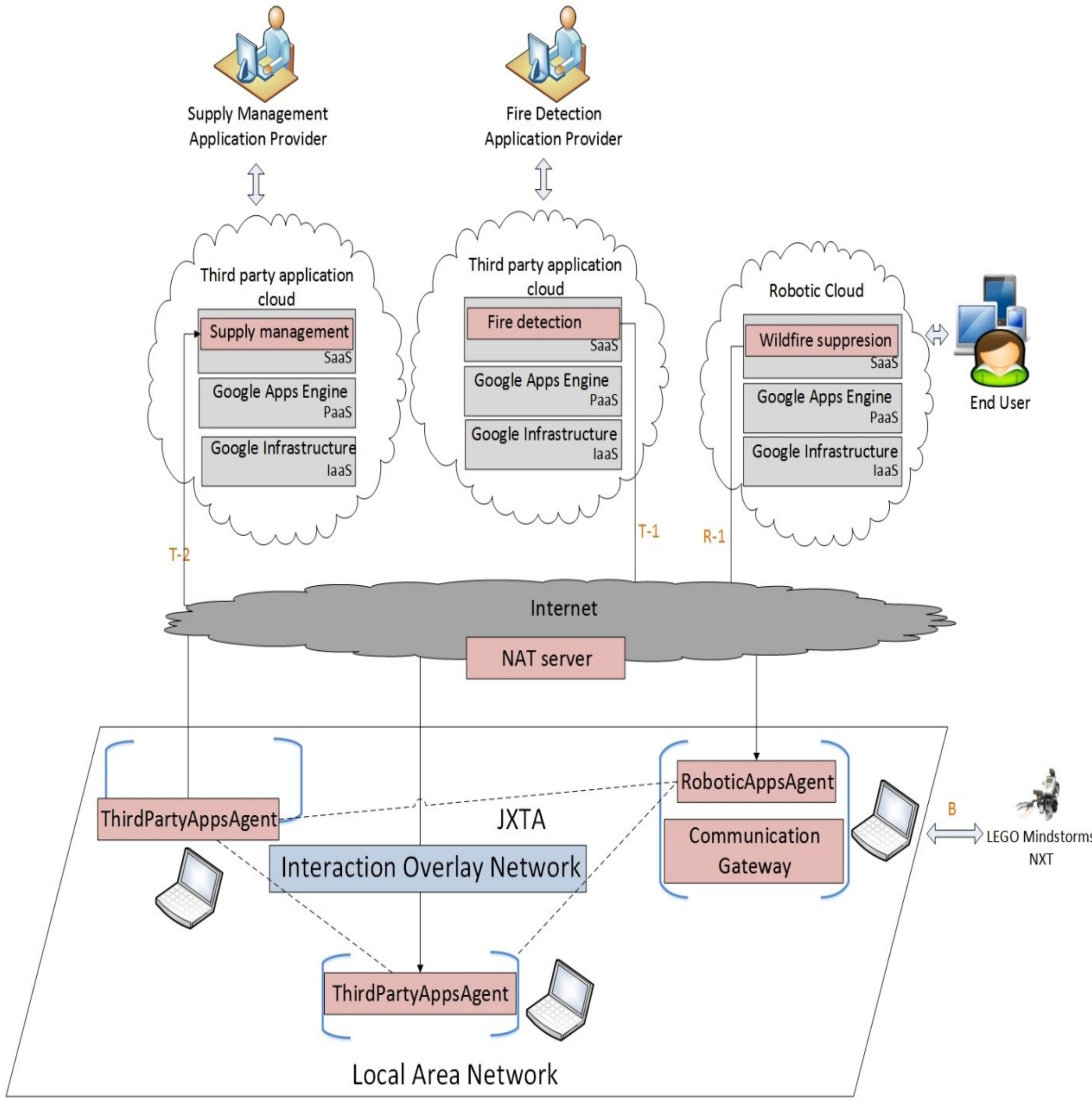

Figure 6-10: The prototype setup

6.3.2 End-to-end Execution of the prototype

The end-to-end execution includes the requests starting from the fire notification to the robot deployment. Figure 6-11 shows a sequence diagram of the end-to-end execution.

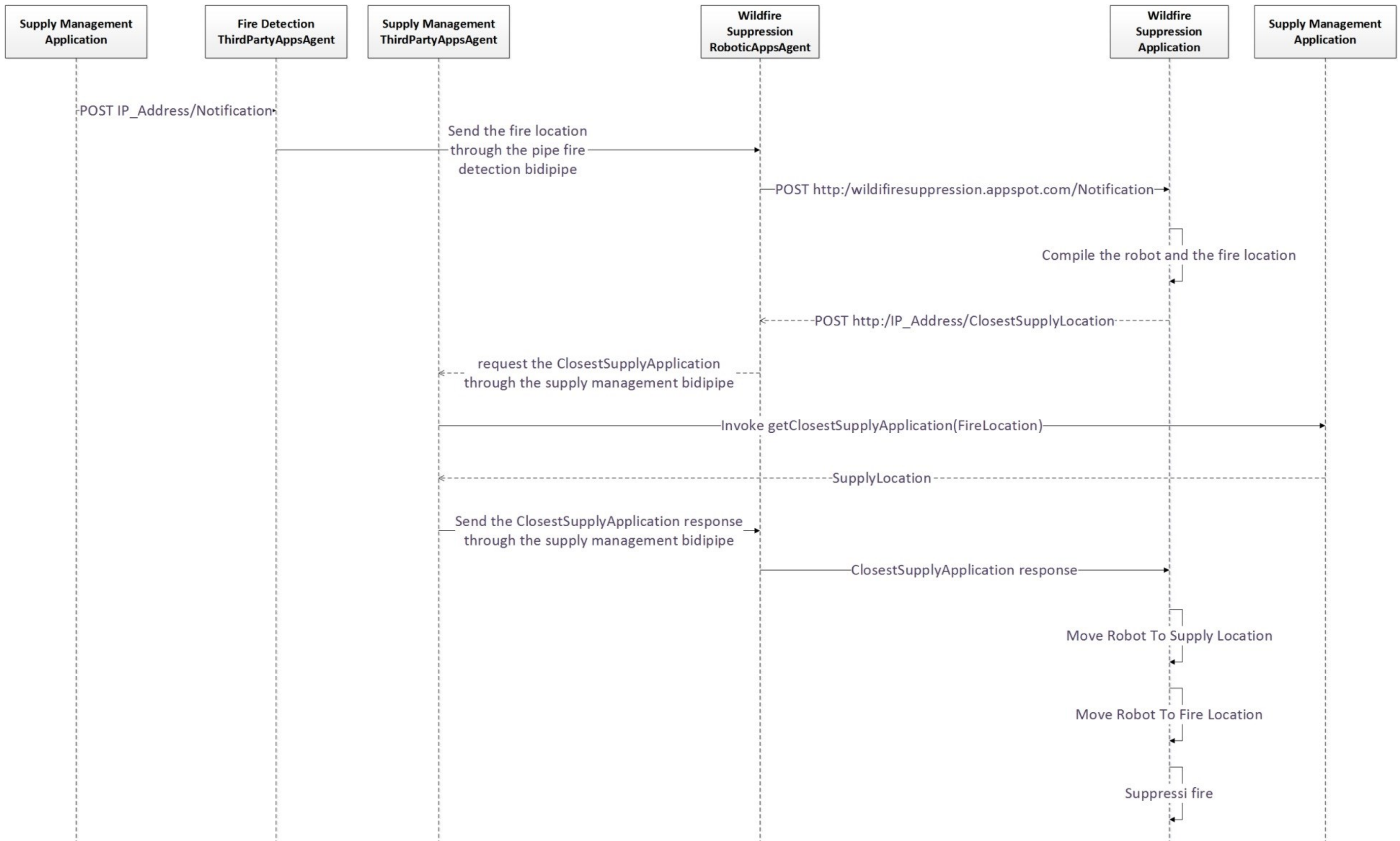

Figure 6-11: The end-to-end execution sequence diagram of the wildfire suppression scenario

6.3.3 Performance Measurements

The main purpose of the measurement is to evaluate the time spent in the overlay has for the discovery and execution procedures.

6.3.3.1 Experimental Setup

The execution setup is the same as the prototype setup. We used three laptops; the first laptop executes the NAT server, the communication gateway application and the *RoboticAppsAgent* node. The second laptop executes the *ThirdPartyAppsAgent* node corresponding to the fire detection application. The third laptop executes *ThirdPartyAppsAgent* node corresponding to the supply management application. The three laptops run with Windows 7 Professional. They have an Intel® Core ™i5-2540 CPU, with 2.60Hz, and 4Gb of RAM. The three laptops are connected through Ethernet cable to a Linksys router; they pertain to the same LAN. The first laptop is also connected to the Internet through its wireless interface.

6.3.3.2 Performance metrics

We evaluated the prototype performance using three metrics: publication, discovery, and execution delays. The publication delay is the time difference between when an application sends a publication request to the overlay and the time it receives a '200OK' response. Two types of publication delays were measured: Fire Detection Publication delay (FDP) and Supply Management Publication delay (SMP). They represent the publication delays for the fire detection and for the supply management applications, respectively.

The discovery delay is the time it takes for the wildfire suppression application to discover third party applications. It is the time difference between when the application sends the discovery request to the overlay (via the *RoboticAppsAgent* node) and the time it gets the description of the available third party applications that fulfill the discovery criteria. This includes the communication delay between the wildfire suppression application and the *RoboticAppsAgent* node and the time it takes to discover the information in the overlay. We call this delay third party application discovery (TPAD). We have also measured separately the discovery delay in the overlay (i.e. the third party applications discovery in the overlay-TPADN) to compare it to the total discovery delay (i.e. TPAD). TPADN represent the part of the discovery procedure that happens in the overlay. It is the time difference between when the *RoboticAppsAgent* node corresponding to the wildfire suppression application sends a discovery request in the overlay and when it receives a response.

We have two types of execution delays, the first is related to the supply management application and the second is related to the fire detection application. For the supply management application execution delay, we measured the supply management end-to-end execution (SME2E), the *ThirdPartyAppsAgent*'s supply management execution delay (TPASM), and the local supply management execution delay (SM). For the fire detection application execution delay, we measured the fire notification end-to-end delay (FDE2E) and the *ThirdPartyAppsAgent*'s fire notification execution delay (TPAFD).

SME2E is the time difference between when the wildfire suppression application sends requests for the closest supply station to the overlay and when it receives the closest supply station information. SME2E includes the TPASM delay. TPASM represents the time spent between when the *ThirdPartyAppsAgent* node invokes the closest supply station execution and when the execution ends. TPASM includes the SM delay. SM represents the time that the supply management application spends for processing the closest supply station. It is calculated by measuring the time the closest supply station method is invoked and when it returns the response.

FDE2E represents the time between when the fire detection sends a fire notification to its node in the overlay, and when the wildfire suppression application receives the notification. FDE2E includes TPAFD. TPAFD is the time difference between when the fire application sends the fire notification to its *ThirdPartyAppsAgent* node and when it receives a '200OK' response.

The total end-to-end (EOE) execution metric is measured from the moment we trigger a fire notification in the fire detection application to the moment when the wildfire suppression application deploys the robot to extinguish the fire.

6.3.3.3 Performance Results

The delays are measured in milliseconds and each result is calculated as an average of 10 experiments.

The average of FDP and SMP is 329ms and 346ms, respectively. We observe that the FDP and SMP average are very close. This is expected because both applications are hosted in Google Infrastructure, and they both use the same cloud PaaS to send requests to the overlay.

Figure 6-12 shows the average time for TPAD (i.e., 377ms), and the average time for TPADN (i.e., 48ms). We can conclude that the time it takes to discover the third party applications in the overlay (i.e., TPADN) is very small (i.e., 1%) compared to the total third party application discovery delay (i.e., TPAD). These results show the viability of using an overlay protocol for discovery.

Figure 6-13 shows SME2E, TPASM, and SM delays. Figure 6-14 shows FDE2E and TPAFD delays. Figure 6-15 shows the TPASM and TPAFD delays.

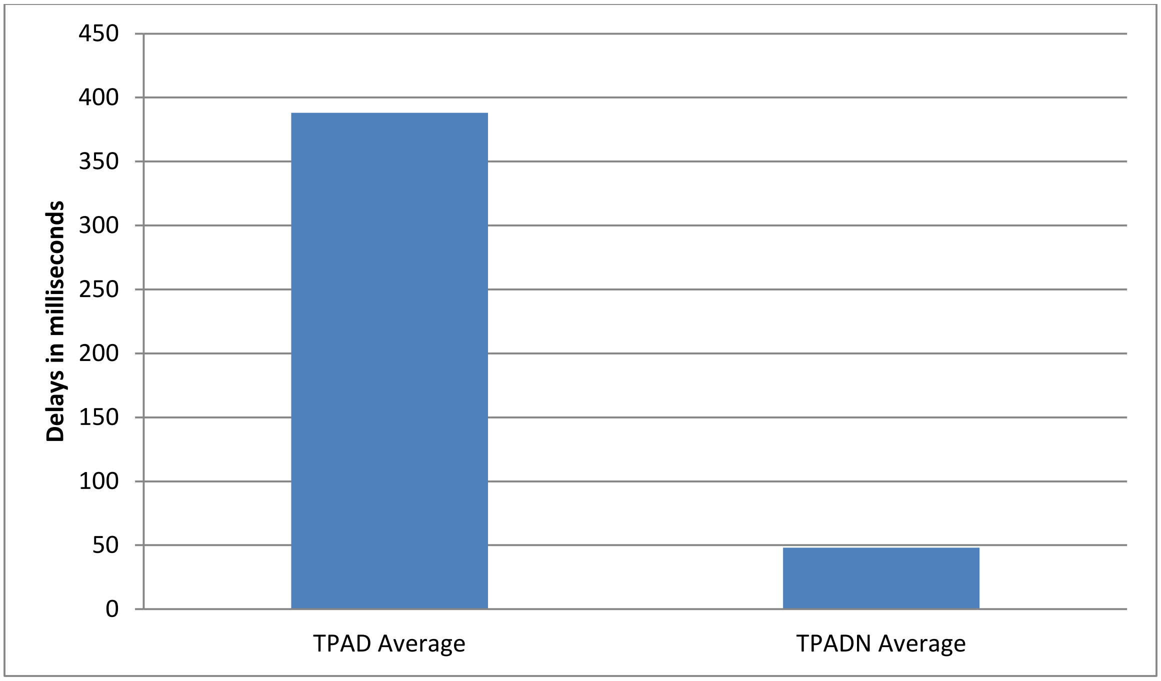

Figure 6-12: Discovery Delays

The average time of SME2E (i.e., 1884ms) is significantly higher than FDE2E (i.e., 737ms). We can observe in Figure 6-11 that TPASM represents most of the time spent in the end-to-end execution of the supply management application (SME2E). The execution time spent by the supply management application for the execution (SM) is negligible compared to TPASM (i.e., SM average is less than 1% of TPASM average). By comparing TPASM and TPAFD in Figure 6-13, we can observe that TPASM is significantly higher than TPAFD (average of TPASM is 1372ms, and the average of TPAFD is 347ms). This is due to the use of SOAP-based interface for the communication between supply management and its corresponding *ThirdPartyAppsAgent* node. Whereas, we use REST-based interface for the communication between the fire detection application and its corresponding *ThirdPartyAppsAgent* node. This conclusion have been mentioned in other studies [32].

The average end-to-end execution time is 2487ms. These results show the viability of using the architecture for fire notification and wildfire suppression.

These results show that for the execution delay can be acceptable for the wildfire suppression application domain. However, such delays may be inappropriate for a surgery-oriented robotic application in a medical setting.

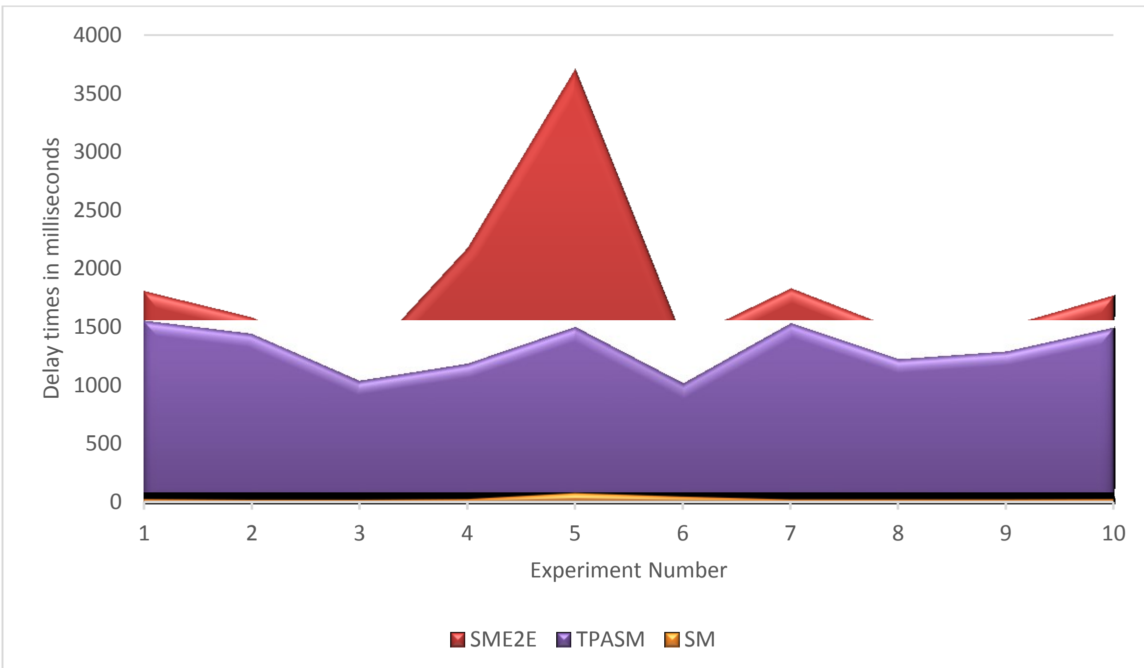

Figure 6-13: Closest Supply Execution Delays

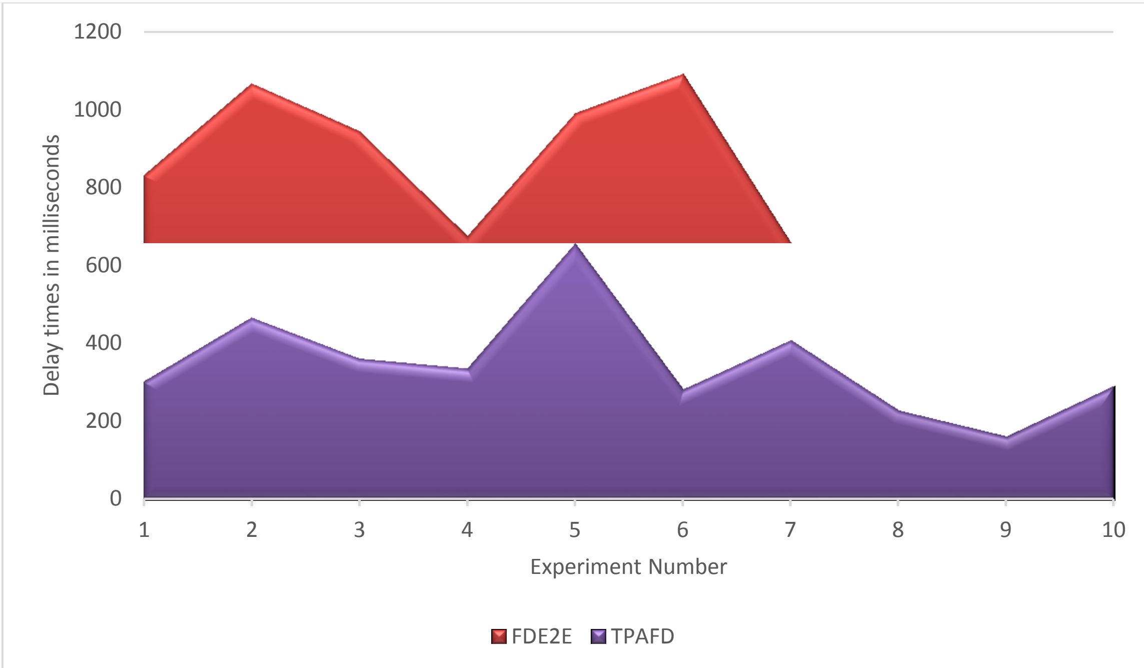

Figure 6-14: Fire Detection Execution Delays

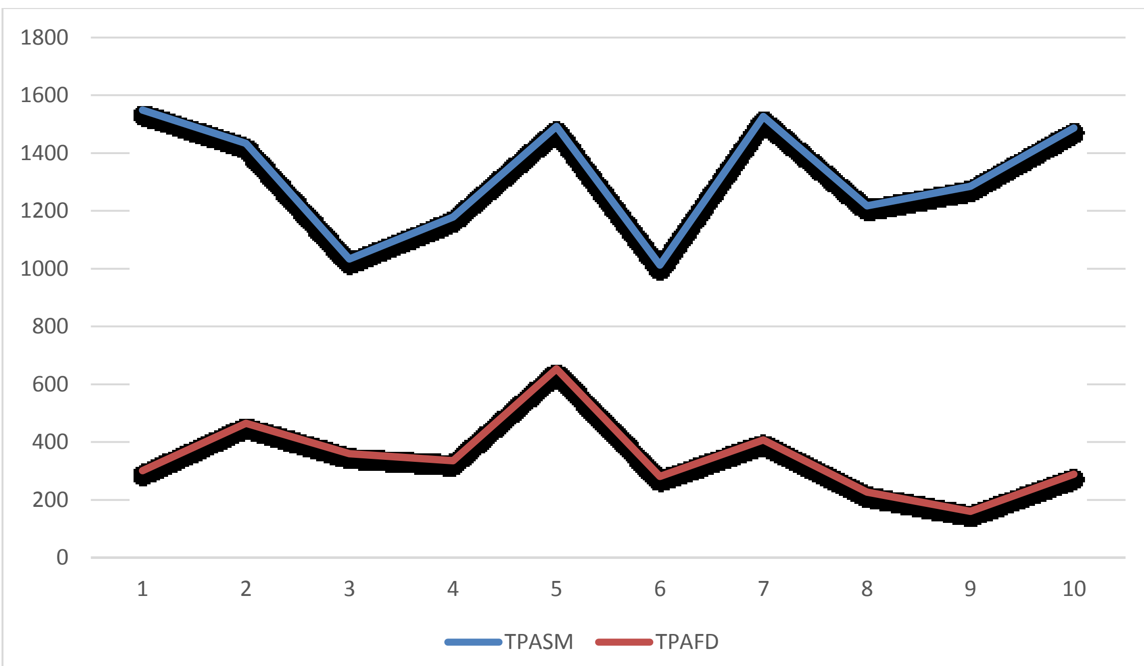

Figure 6-15: TPAFD and TPASM delays

6.4 Chapter Summary

In this chapter, we described the overall prototype architecture, including the software tools we used in the implementation. Then, we presented the prototype architecture in details. We described the application we implemented (i.e., wildfire suppression, the fire detection, and the supply management applications). Next, we described the interaction overlay network in details. Then we presented the prototype setup, and the performance measurements.

7 Chapter 7

Conclusion and Research Directions

In this chapter we will summarize the thesis contributions. Then, we will identify some research directions.

7.1 Contributions Summary

Robotic applications are near ubiquitous. Unfortunately, provisioning them in a cost efficient manner remains a difficult task. Third party applications are hardly re-used when developing robotic applications. Cloud computing is a promising new paradigm for application provisioning. It represents an interesting candidate for robotic application provisioning.

This thesis proposed a novel business model that tackles the issue of robotic applications and third party application clouds for cost efficient robotic application provisioning. We used the provisioning lifecycle to derive the requirements. Among the requirements, we identified general requirements, development-related requirements, deployment-related requirements, and operation-related requirements.

The business model comprises four categories of actors: the end-users, the robotic cloud providers, the third party cloud providers and the cloud interaction framework provider. The role of the interaction framework provider is to offer a platform that facilitates the interactions between the third party application cloud providers and the robotic application cloud providers. We presented derived the interaction framework architecture based on the business model. This architecture uses P2P overlay networks as a cornerstone.

We have built a prototype based on the wildfire scenario. We implemented a cloud robotic application that was hosted on Google Infrastructure, two third party applications and the interaction overlay network. Performance measurement were calculated and analyzed. Performance results show that further research work must be done in the case of application domains, such as robot-assisted surgery.

7.2 Research Directions

Several research items need to be addressed. First, we will present the first research item which is the application of cloud paradigm to robotic applications at the IaaS, PaaS and SaaS level. Then we will describe research directions related to the third party application provisioning.

7.2.1 Application of cloud paradigm to robotic applications and service provisioning

Several research items need to be addressed at the IaaS, PaaS and SaaS level. At the SaaS level, the main challenge concerns the interaction between the robotic application offered as SaaS and the end-users. The interface should be lightweight enough to allow small footprint devices to access the robotic application.

Also, one of the aspects of PaaS in robotic clouds is to take into account the robotic application peculiarities, which are not supported by today's existing PaaS. Another challenge is that PaaS should cater to both experienced and novice developer needs.

Robot virtualization is the key research area at the IaaS level. Research on robot virtualization is at its very early stage. Some of the challenges that we identified in this papers is the need for physical presence to be factored in when virtualizing the robots. In the wildfire management scenario, for instance, it would not make sense to assign a ground robot to distinguish a fire in a remote area, even if the robot is idle. A robot currently present in the fire scene would be a better choice. Furthermore, algorithms are needed to select the most appropriate robots for a given task. The algorithms should minimize the number of robots to use for each task, minimize the distance between the fire scene and the current location of the selected robots, and maximize the number of tasks that can be executed in parallel. In the wildfire management scenario for example, it would be more appropriate to assign robots in a way that leaves room to react if a fire takes place in a different zone.

Another challenge is to define the technology interfaces between the IaaS, the PaaS, and the SaaS providers in both the robotic cloud and the third party cloud as presented in the business model.

7.2.2 Third party application ontology provisioning

Among the challenges related to the ontology, we have the design, integration and the provisioning. For the ontology design, the challenge is to build well-defined ontologies for the robotic applications and related third party domains. In the wildfire suppression scenario, for instance, an ontology is needed to define the concepts related to the fire detection, suppression and supply management. These different ontologies should be designed in a way that facilitates their integration in order to give the robotic application a global and unified view of all the third party applications offered in the overlay. For ontology provisioning, the challenge is to make it possible for the various agents to learn about new concepts and to make the provisioning distributed to avoid the *OntologyStore* node from becoming a bottleneck in the architecture.

Bibliography

1. L. M. Vaquero, L. Rodero-Merino, J. Caceres and M. Lindner, "A break in the clouds " *ACM SIGCOMM Computer Communication Review,* vol. 39, pp. 50, 2008

2. B. P. Rimal, E. Choi and I. Lumb, "2009 fifth international joint conference on INC, IMS and IDC; A taxonomy and survey of cloud computing systems" in 2009, pp. 44-51.

3. Seungbin Moon, Soon-Geul Lee and Kwang-Ho Park, "Recent progress of robotic vocabulary standardization efforts in ISO," in *SICE Annual Conference 2010, Proceedings of,* 2010, pp. 266-268.

4. Q. Zhang, L. Cheng and R. Boutaba, "Cloud computing: state-of-the-art and research challenges," Journal of Internet Services and Applications, vol. 1, pp. 7-18, 2010.

5. "Amazon Elastic Compute Cloud," http://aws.amazon.com/ec2/.

6. "GoGrid cloud hosting," http://www.gogrid.com.

7. "Google Apps Engine," https://developers.google.com/appengine/.

8. "Windows Azure," www.microsoft.com/azure.

9. "Salesforce," http://www.salesforce.com/platform.

10. "SAP Business ByDesign," https://www54.sap.com/pc/tech/cloud/software/business-management-bydesign/overview/index.html.

11. J. Sahoo, S. Mohapatra and R. Lath, "Virtualization: A survey on concepts, taxonomy and associated security issues," in Computer and Network Technology (ICCNT), 2010 Second International Conference on, 2010, pp. 222-226.

12. R. A. Beasley, "Medical robots: current systems and research directions," Journal of Robotics, vol. 2012, 2012.

13. K. Niechwiadowicz and Z. Khan, "Robot based logistics system for hospitals-survey," in IDT Workshop on Interesting Results in Computer Science and Engineering, 2008,

14. L. Pedersen, D. Kortenkamp, D. Wettergreen and I. Nourbakhsh, "A survey of space robotics," in Proceedings of the 7th International Symposium on Artificial Intelligence, Robotics and Automation in Space, 2003, pp. 19-23.

15. A. Chikwanha, S. Motepe and R. Stopforth, "Survey and requirements for search and rescue ground and air vehicles for mining applications," in Mechatronics and Machine Vision in Practice (M2VIP), 2012 19th International Conference, 2012, pp. 105-109.

16. A. Osterwalder, Y. Pigneur and C. L. Tucci, "Clarifying business models: Origins, present, and future of the concept," Communications of the Association for Information Systems, vol. 16, pp. 1-25, 2005.

17. J. Jackson, "Microsoft robotics studio: A technical introduction " IEEE Robotics & Automation Magazine, vol. 14, pp. 82-87, 2007.

18. M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler and A. Y. Ng, "ROS: An open-source robot operating system," in ICRA Workshop on Open Source Software, 2009, .

19. J. Ding, I. Balasingham and P. Bouvry, "Management of overlay networks: A survey," in Mobile Ubiquitous Computing, Systems, Services and Technologies, 2009. UBICOMM'09. Third International Conference on, 2009, pp. 249-255.

20. E. K. Lua, J. Crowcroft, M. Pias, R. Sharma and S. Lim, "A survey and comparison of peer-to-peer overlay network schemes." IEEE Communications Surveys and Tutorials, vol. 7, pp. 72-93, 2005.

21. Y. Chen, Z. Du and M. García-Acosta, "2010 fifth IEEE international symposium on service oriented system engineering; robot as a service in cloud computing," in 2010, pp. 151-158.

22. M. P. Papazoglou and W. Heuvel, "Service oriented architectures: approaches, technologies and research issues " The VLDB Journal, vol. 16, pp. 389-415, 2007; 2007.

23. Z. Du, W. Yang, Y. Chen, X. Sun, X. Wang and C. Xu, "2011 tenth international symposium on autonomous decentralized systems; design of a robot cloud center," in 2011, pp. 269-275.

24. R. Arumugam, V. R. Enti, Liu Bingbing, Wu Xiaojun, K. Baskaran, Foong Foo Kong, A. S. Kumar, Kang Dee Meng and Goh Wai Kit, "DAvinCi: A cloud computing framework for service robots," in Robotics and Automation (ICRA), 2010 IEEE International Conference on, 2010, pp. 3084-3089.

25. AiLing Duan, "Research and application of distributed parallel search hadoop algorithm," in Systems and Informatics (ICSAI), 2012 International Conference on, 2012, pp. 2462-2465.

26. M. Waibel, M. Beetz, J. Civera, R. D'Andrea, J. Elfring, D. Galvez-Lopez, K. Haussermann, R. Janssen, J. M. M. Montiel, A. Perzylo, B. Schiessle, M. Tenorth, O. Zweigle and R. van de Molengraft, "RoboEarth," Robotics & Automation Magazine, IEEE, vol. 18, pp. 69-82, 2011.

27. Yan-You Chen, Jhing-Fa Wang, Po-Chuan Lin, Po-Yi Shih, Hsin-Chun Tsai and Da-Yu Kwan, "Human-robot interaction based on cloud computing infrastructure for senior companion," in TENCON 2011 - 2011 IEEE Region 10 Conference, 2011, pp. 1431-1434.

28. H. Bistry and Jianwei Zhang, "A cloud computing approach to complex robot vision tasks using smart camera systems," in Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on, 2010, pp. 3195-3200.

29. R. Tsuchiya, S. Shimazaki, T. Sakai, S. Terada, K. Igarashi, D. Hanawa and K. Oguchi, "Simulation environment based on smartphones for cloud computing robots," in Telecommunications and Signal Processing (TSP), 2012 35th International Conference on, 2012, pp. 96-100.

30. K. K. Breitman and J. C. Sampaio do Prado Leite, "Ontology as a requirements engineering product," in Requirements Engineering Conference, 2003. Proceedings. 11th IEEE International, 2003, pp. 309-319.

31. I. Houidi, M. Mechtri, W. Louati and D. Zeghlache, "Cloud service delivery across multiple cloud platforms," in Services Computing (SCC), 2011 IEEE International Conference on, 2011, pp. 741-742.

32. F. Belqasmi, J. Singh, S. Y. Bani Melhem and R. H. Glitho, "SOAP-Based vs. RESTful Web Services: A Case Study for Multimedia Conferencing," Internet Computing, IEEE, vol. 16, pp. 54-63, 2012.

33. Chen Shan, Chang Heng and Zou Xianjun, "Inter-cloud operations via NGSON," Communications Magazine, IEEE, vol. 50, pp. 82-89, 2012.

34. N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker and J. Turner, "OpenFlow: enabling innovation in campus networks," ACM SIGCOMM Computer Communication Review, vol. 38, pp. 69-74, 2008.

35. Kalabokidis, K., N. Athanasis, and M. Vaitis. "OntoFire: an ontology-based geo-portal for wildfires." Natural Hazards and Earth System Science 11.12 (2011): 3157-3170.

36. L. Gong, "JXTA: A network programming environment," Internet Computing, IEEE, vol. 5, pp. 88-95, 2001.

37. Zahariev, Alexander. "Google app engine" TKK T-110.5190 Seminar on Internetworking. 2009.

38. S. H. Kim and J. W. Jeon, "Introduction for freshmen to embedded systems using LEGO Mindstorms," Education, IEEE Transactions on, vol. 52, pp. 99-108, 2009.

39. F. Belqasmi, R. Glitho and Chunyan Fu, "RESTful web services for service provisioning in next-generation networks: a survey," Communications Magazine, IEEE, vol. 49, pp. 66-73, 2011.

40. H. Li, "RESTful web service frameworks in java," in Signal Processing, Communications and Computing (ICSPCC), 2011 IEEE International Conference on, 2011, pp. 1-4.

41. S. Susila, S. Vadivel and A. Julka, "Broker architecture for web service selection using SOAPUI," in Cloud Computing Technologies, Applications and Management (ICCCTAM), 2012 International Conference on, 2012, pp. 219-222.

42. M. Jacobs and P. Leydekkers, "Specification of synchronization in multimedia conferencing services using the TINA life-cycle model," in Services in Distributed and Networked Environments, 1995., Second International Workshop on, 1995, pp. 43-50del